Method Descriptions

Theme **get_default_theme**()

Returns a reference to the default engine Theme. This theme resource is responsible for the out-of-the-box look of Control nodes and cannot be overridden.

Theme **get_project_theme**()

Returns a reference to the custom project Theme. This theme resources allows to override the default engine theme for every control node in the project.

To set the project theme, see ProjectSettings.gui/theme/custom.

Thread

Inherits: RefCounted **<** Object

A unit of execution in a process.

Description

A unit of execution in a process. Can run methods on Objects simultaneously. The use of synchronization via Mutex or Semaphore is advised if working with shared objects.

Warning:

To ensure proper cleanup without crashes or deadlocks, when a **Thread**'s reference count reaches zero and it is therefore destroyed, the following conditions must be met:

- It must not have any Mutex objects locked.
- It must not be waiting on any Semaphore objects.
- wait_to_finish should have been called on it.

Tutorials

- Using multiple threads
- Thread-safe APIs
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

Methods

String	get_id() const
bool	is_alive() const
bool	is_started() const
void	set_thread_safety_checks_enabled(enabled: bool) static
Error	start(callable: Callable, priority: Priority = 1)
Variant	wait_to_finish()

Enumerations

enum **Priority**: 🔗

Priority **PRIORITY_LOW** = `0`

A thread running with lower priority than normally.

Priority **PRIORITY_NORMAL** = `1`

A thread with a standard priority.

Priority **PRIORITY_HIGH** = `2`

A thread running with higher priority than normally.

Method Descriptions

String **get_id**() const 🔗

Returns the current **Thread**'s ID, uniquely identifying it among all threads. If the **Thread** has not started running or if wait_to_finish has been called, this returns an empty string.

---

bool **is_alive**() const 🔗

Returns `true` if this **Thread** is currently running the provided function. This is useful for determining if wait_to_finish can be called without blocking the calling thread.

To check if a **Thread** is joinable, use is_started.

---

bool **is_started**() const 🔗

Returns `true` if this **Thread** has been started. Once started, this will return `true` until it is joined using wait_to_finish. For checking if a **Thread** is still executing its task, use is_alive.

---

void **set_thread_safety_checks_enabled**(enabled: bool) static 🔗

Sets whether the thread safety checks the engine normally performs in methods of certain classes (e.g., Node) should happen **on the current thread**.

The default, for every thread, is that they are enabled (as if called with `enabled` being `true`).

Those checks are conservative. That means that they will only succeed in considering a call thread-safe (and therefore allow it to happen) if the engine can guarantee such safety.

Because of that, there may be cases where the user may want to disable them (`enabled` being `false`) to make certain operations allowed again. By doing so, it becomes the user's responsibility to ensure thread safety (e.g., by using Mutex) for those objects that are otherwise protected by the engine.

Note: This is an advanced usage of the engine. You are advised to use it only if you know what you are doing and there is no safer way.

Note: This is useful for scripts running on either arbitrary **Thread** objects or tasks submitted to the WorkerThreadPool. It doesn't apply to code running during Node group processing, where the checks will be always performed.

Note: Even in the case of having disabled the checks in a WorkerThreadPool task, there's no need to re-enable them at the end. The engine will do so.

Error **start**(callable: Callable, priority: Priority = 1) 🔗

Starts a new **Thread** that calls `callable`.

If the method takes some arguments, you can pass them using Callable.bind.

The `priority` of the **Thread** can be changed by passing a value from the Priority enum.

Returns @GlobalScope.OK on success, or @GlobalScope.ERR_CANT_CREATE on failure.

Variant **wait_to_finish**() 🔗

Joins the **Thread** and waits for it to finish. Returns the output of the Callable passed to start.

Should either be used when you want to retrieve the value returned from the method called by the **Thread** or before freeing the instance that contains the **Thread**.

To determine if this can be called without blocking the calling thread, check if is_alive is `false`.

TileData

Inherits: Object

Settings for a single tile in a TileSet.

Description

TileData object represents a single tile in a TileSet. It is usually edited using the tileset editor, but it can be modified at runtime using TileMap._tile_data_runtime_update.

Properties

bool	flip_h	`false`
bool	flip_v	`false`
Material	material	
Color	modulate	`Color(1, 1, 1, 1)`
float	probability	`1.0`
int	terrain	`-1`
int	terrain_set	`-1`

Vector2i	texture_origin	`Vector2i(0, 0)`
bool	transpose	`false`
int	y_sort_origin	`0`
int	z_index	`0`

Methods

void	add_collision_polygon(layer_id: int)
float	get_collision_polygon_one_way_margin(layer_id: int, polygon_index: int) const
PackedVector2Array	get_collision_polygon_points(layer_id: int, polygon_index: int) const
int	get_collision_polygons_count(layer_id: int) const
float	get_constant_angular_velocity(layer_id: int) const
Vector2	get_constant_linear_velocity(layer_id: int) const

Variant	get_custom_data(layer_name: String) const
Variant	get_custom_data_by_layer_id(layer_id: int) const
NavigationPolygon	get_navigation_polygon(layer_id: int, flip_h: bool = false, flip_v: bool = false, transpose: bool = false) const
OccluderPolygon2D	get_occluder(layer_id: int, flip_h: bool = false, flip_v: bool = false, transpose: bool = false) const
int	get_terrain_peering_bit(peering_bit: CellNeighbor) const
bool	is_collision_polygon_one_way(layer_id: int, polygon_index: int) const
bool	is_valid_terrain_peering_bit(peering_bit: CellNeighbor) const
void	remove_collision_polygon(layer_id: int, polygon_index: int)
void	set_collision_polygon_one_way(layer_id: int, polygon_index: int, one_way: bool)

void	set_collision_polygon_one_way_margin(layer_id: int, polygon_index: int, one_way_margin: float)
void	set_collision_polygon_points(layer_id: int, polygon_index: int, polygon: PackedVector2Array)
void	set_collision_polygons_count(layer_id: int, polygons_count: int)
void	set_constant_angular_velocity(layer_id: int, velocity: float)
void	set_constant_linear_velocity(layer_id: int, velocity: Vector2)
void	set_custom_data(layer_name: String, value: Variant)
void	set_custom_data_by_layer_id(layer_id: int, value: Variant)
void	set_navigation_polygon(layer_id: int, navigation_polygon: NavigationPolygon)
void	set_occluder(layer_id: int, occluder_polygon: OccluderPolygon2D)

void	set_terrain_peering_bit(peering_bit: CellNeighbor, terrain: int)

Signals

changed()

Emitted when any of the properties are changed.

Property Descriptions

bool **flip_h** = `false`

- void **set_flip_h**(value: bool)
- bool **get_flip_h**()

If `true`, the tile will have its texture flipped horizontally.

bool **flip_v** = `false`

- void **set_flip_v**(value: bool)
- bool **get_flip_v**()

If `true`, the tile will have its texture flipped vertically.

Material **material**

- void **set_material**(value: Material)
- Material **get_material**()

The Material to use for this **TileData**. This can be a CanvasItemMaterial to use the default shader, or a ShaderMaterial to use a custom shader.

Color **modulate** = `Color(1, 1, 1, 1)`

- void **set_modulate**(value: Color)
- Color **get_modulate**()

Color modulation of the tile.

float **probability** = `1.0`

- void **set_probability**(value: float)
- float **get_probability**()

Relative probability of this tile being selected when drawing a pattern of random tiles.

int **terrain** = `-1`

- void **set_terrain**(value: int)
- int **get_terrain**()

ID of the terrain from the terrain set that the tile uses.

int **terrain_set** = `-1`

- void **set_terrain_set**(value: int)
- int **get_terrain_set**()

ID of the terrain set that the tile uses.

Vector2i **texture_origin** = `Vector2i(0, 0)`

- void **set_texture_origin**(value: Vector2i)
- Vector2i **get_texture_origin**()

Offsets the position of where the tile is drawn.

---

bool **transpose** = `false`

- void **set_transpose**(value: bool)
- bool **get_transpose**()

If `true`, the tile will display transposed, i.e. with horizontal and vertical texture UVs swapped.

---

int **y_sort_origin** = `0`

- void **set_y_sort_origin**(value: int)
- int **get_y_sort_origin**()

Vertical point of the tile used for determining y-sorted order.

---

int **z_index** = `0`

- void **set_z_index**(value: int)
- int **get_z_index**()

Ordering index of this tile, relative to TileMap.

Method Descriptions

void **add_collision_polygon**(layer_id: int)

Adds a collision polygon to the tile on the given TileSet physics layer.

---

float **get_collision_polygon_one_way_margin**(layer_id: int, polygon_index: int) const 🔗

Returns the one-way margin (for one-way platforms) of the polygon at index `polygon_index` for TileSet physics layer with index `layer_id`.

PackedVector2Array **get_collision_polygon_points**(layer_id: int, polygon_index: int) const 🔗

Returns the points of the polygon at index `polygon_index` for TileSet physics layer with index `layer_id`.

---

int **get_collision_polygons_count**(layer_id: int) const 🔗

Returns how many polygons the tile has for TileSet physics layer with index `layer_id`.

---

float **get_constant_angular_velocity**(layer_id: int) const 🔗

Returns the constant angular velocity applied to objects colliding with this tile.

Vector2 **get_constant_linear_velocity**(layer_id: int) const 🔗

Returns the constant linear velocity applied to objects colliding with this tile.

Variant **get_custom_data**(layer_name: String) const 🔗

Returns the custom data value for custom data layer named `layer_name`.

Variant **get_custom_data_by_layer_id**(layer_id: int) const 🔗

Returns the custom data value for custom data layer with index `layer_id`.

NavigationPolygon **get_navigation_polygon**(layer_id: int, flip_h: bool = false, flip_v: bool = false, transpose: bool = false) const 🔗

Returns the navigation polygon of the tile for the TileSet navigation layer with index `layer_id`.

`flip_h`, `flip_v`, and `transpose` allow transforming the returned polygon.

OccluderPolygon2D **get_occluder**(layer_id: int, flip_h: bool = false, flip_v: bool = false, transpose: bool = false) const 🔗

Returns the occluder polygon of the tile for the TileSet occlusion layer with index `layer_id`.

`flip_h`, `flip_v`, and `transpose` allow transforming the returned polygon.

---

int **get_terrain_peering_bit**(peering_bit: CellNeighbor) const 🔗

Returns the tile's terrain bit for the given `peering_bit` direction. To check that a direction is valid, use is_valid_terrain_peering_bit.

---

bool **is_collision_polygon_one_way**(layer_id: int, polygon_index: int) const

Returns whether one-way collisions are enabled for the polygon at index `polygon_index` for TileSet physics layer with index `layer_id`.

---

bool **is_valid_terrain_peering_bit**(peering_bit: CellNeighbor) const

Returns whether the given `peering_bit` direction is valid for this tile.

---

void **remove_collision_polygon**(layer_id: int, polygon_index: int)

Removes the polygon at index `polygon_index` for TileSet physics layer with index `layer_id`.

---

void **set_collision_polygon_one_way**(layer_id: int, polygon_index: int, one_way: bool)

Enables/disables one-way collisions on the polygon at index `polygon_index` for TileSet physics layer with index `layer_id`.

---

void **set_collision_polygon_one_way_margin**(layer_id: int, polygon_index: int, one_way_margin: float)

Enables/disables one-way collisions on the polygon at index `polygon_index` for TileSet physics layer with index `layer_id`.

---

void **set_collision_polygon_points**(layer_id: int, polygon_index: int, polygon: PackedVector2Array)

Sets the points of the polygon at index `polygon_index` for TileSet physics layer with index `layer_id`.

---

void **set_collision_polygons_count**(layer_id: int, polygons_count: int)

Sets the polygons count for TileSet physics layer with index `layer_id`.

---

void **set_constant_angular_velocity**(layer_id: int, velocity: float)

Sets the constant angular velocity. This does not rotate the tile. This angular velocity is applied to objects colliding with this tile.

---

void **set_constant_linear_velocity**(layer_id: int, velocity: Vector2)

Sets the constant linear velocity. This does not move the tile. This linear velocity is applied to objects colliding with this tile. This is useful to create conveyor belts.

---

void **set_custom_data**(layer_name: String, value: Variant)

Sets the tile's custom data value for the TileSet custom data layer with name `layer_name`.

---

void **set_custom_data_by_layer_id**(layer_id: int, value: Variant)

Sets the tile's custom data value for the TileSet custom data layer with index `layer_id`.

---

void **set_navigation_polygon**(layer_id: int, navigation_polygon: NavigationPolygon) 🔗

Sets the navigation polygon for the TileSet navigation layer with index `layer_id`.

---

void **set_occluder**(layer_id: int, occluder_polygon: OccluderPolygon2D) 🔗

Sets the occluder for the TileSet occlusion layer with index `layer_id`.

---

void **set_terrain_peering_bit**(peering_bit: CellNeighbor, terrain: int) 🔗

Sets the tile's terrain bit for the given `peering_bit` direction. To check that a direction is valid, use is_valid_terrain_peering_bit.

Time

Inherits: Object

A singleton for working with time data.

Description

The Time singleton allows converting time between various formats and also getting time information from the system.

This class conforms with as many of the ISO 8601 standards as possible. All dates follow the Proleptic Gregorian calendar. As such, the day before `1582-10-15` is `1582-10-14`, not `1582-10-04`. The year before 1 AD (aka 1 BC) is number `0`, with the year before that (2 BC) being `-1`, etc.

Conversion methods assume "the same timezone", and do not handle timezone conversions or DST automatically. Leap seconds are also not handled, they must be done manually if desired. Suffixes such as "Z" are not handled, you need to strip them away manually.

When getting time information from the system, the time can either be in the local timezone or UTC depending on the `utc` parameter. However, the get_unix_time_from_system method always uses UTC as it returns the seconds passed since the Unix epoch [https://en.wikipedia.org/wiki/Unix_time].

Important: The `_from_system` methods use the system clock that the user can manually set. **Never use** this method for precise time calculation since its results are subject to automatic adjustments by the user or the operating system. **Always use** get_ticks_usec or get_ticks_msec for precise time calculation instead, since they are guaranteed to be monotonic (i.e. never decrease).

Methods

Dictionary	get_date_dict_from_system(utc: bool = false) const
Dictionary	get_date_dict_from_unix_time(unix_time_val: int) const
String	get_date_string_from_system(utc: bool = false) const
String	get_date_string_from_unix_time(unix_time_val: int) const
Dictionary	get_datetime_dict_from_datetime_string(datetime: String, weekday: bool) const
Dictionary	get_datetime_dict_from_system(utc: bool = false) const
Dictionary	get_datetime_dict_from_unix_time(unix_time_val: int) const
String	get_datetime_string_from_datetime_dict(datetime: Dictionary, use_space: bool) const

String	get_datetime_string_from_system(utc: bool = false, use_space: bool = false) const
String	get_datetime_string_from_unix_time(unix_time_val: int, use_space: bool = false) const
String	get_offset_string_from_offset_minutes(offset_minutes: int) const
int	get_ticks_msec() const
int	get_ticks_usec() const
Dictionary	get_time_dict_from_system(utc: bool = false) const
Dictionary	get_time_dict_from_unix_time(unix_time_val: int) const
String	get_time_string_from_system(utc: bool = false) const
String	get_time_string_from_unix_time(unix_time_val: int) const
Dictionary	get_time_zone_from_system() const
int	get_unix_time_from_datetime_dict(datetime: Dictionary) const

int	get_unix_time_from_datetime_string(datetime: String) const
float	get_unix_time_from_system() const

Enumerations

enum **Month**:

Month **MONTH_JANUARY** = `1`

The month of January, represented numerically as `01`.

Month **MONTH_FEBRUARY** = `2`

The month of February, represented numerically as `02`.

Month **MONTH_MARCH** = `3`

The month of March, represented numerically as `03`.

Month **MONTH_APRIL** = `4`

The month of April, represented numerically as `04`.

Month **MONTH_MAY** = `5`

The month of May, represented numerically as `05`.

Month **MONTH_JUNE** = `6`

The month of June, represented numerically as `06`.

Month **MONTH_JULY** = `7`

The month of July, represented numerically as `07`.

Month **MONTH_AUGUST** = `8`

The month of August, represented numerically as `08`.

Month **MONTH_SEPTEMBER** = `9`

The month of September, represented numerically as `09`.

Month **MONTH_OCTOBER** = `10`

The month of October, represented numerically as `10`.

Month **MONTH_NOVEMBER** = `11`

The month of November, represented numerically as `11`.

Month **MONTH_DECEMBER** = `12`

The month of December, represented numerically as `12`.

---

enum **Weekday**: 🔗

Weekday **WEEKDAY_SUNDAY** = `0`

The day of the week Sunday, represented numerically as `0`.

Weekday **WEEKDAY_MONDAY** = `1`

The day of the week Monday, represented numerically as `1`.

Weekday **WEEKDAY_TUESDAY** = `2`

The day of the week Tuesday, represented numerically as `2`.

Weekday **WEEKDAY_WEDNESDAY** = `3`

The day of the week Wednesday, represented numerically as `3`.

Weekday **WEEKDAY_THURSDAY** = `4`

The day of the week Thursday, represented numerically as `4`.

Weekday **WEEKDAY_FRIDAY** = `5`

The day of the week Friday, represented numerically as `5`.

Weekday **WEEKDAY_SATURDAY** = `6`

The day of the week Saturday, represented numerically as `6`.

Method Descriptions

Dictionary **get_date_dict_from_system**(utc: bool = false) const 🔗

Returns the current date as a dictionary of keys: `year`, `month`, `day`, and `weekday`.

The returned values are in the system's local time when `utc` is `false`, otherwise they are in UTC.

Dictionary **get_date_dict_from_unix_time**(unix_time_val: int) const 🔗

Converts the given Unix timestamp to a dictionary of keys: `year`, `month`, `day`, and `weekday`.

String **get_date_string_from_system**(utc: bool = false) const

Returns the current date as an ISO 8601 date string (YYYY-MM-DD).

The returned values are in the system's local time when `utc` is `false`, otherwise they are in UTC.

String **get_date_string_from_unix_time**(unix_time_val: int) const

Converts the given Unix timestamp to an ISO 8601 date string (YYYY-MM-DD).

Dictionary **get_datetime_dict_from_datetime_string**(datetime: String, weekday: bool) const

Converts the given ISO 8601 date and time string (YYYY-MM-DDTHH:MM:SS) to a dictionary of keys: `year`, `month`, `day`, `weekday`, `hour`, `minute`, and `second`.

If `weekday` is `false`, then the `weekday` entry is excluded (the calculation is relatively expensive).

Note: Any decimal fraction in the time string will be ignored silently.

Dictionary **get_datetime_dict_from_system**(utc: bool = false) const

Returns the current date as a dictionary of keys: `year`, `month`, `day`, `weekday`, `hour`, `minute`, `second`, and `dst` (Daylight Savings Time).

Dictionary **get_datetime_dict_from_unix_time**(unix_time_val: int) const 🔗

Converts the given Unix timestamp to a dictionary of keys: `year`, `month`, `day`, `weekday`, `hour`, `minute`, and `second`.

The returned Dictionary's values will be the same as the get_datetime_dict_from_system if the Unix timestamp is the current time, with the exception of Daylight Savings Time as it cannot be determined from the epoch.

String **get_datetime_string_from_datetime_dict**(datetime: Dictionary, use_space: bool) const 🔗

Converts the given dictionary of keys to an ISO 8601 date and time string (YYYY-MM-DDTHH:MM:SS).

The given dictionary can be populated with the following keys: `year`, `month`, `day`, `hour`, `minute`, and `second`. Any other entries (including `dst`) are ignored.

If the dictionary is empty, `0` is returned. If some keys are omitted, they default to the equivalent values for the Unix epoch timestamp 0 (1970-01-01 at 00:00:00).

If `use_space` is `true`, the date and time bits are separated by an empty space character instead of the letter T.

String **get_datetime_string_from_system**(utc: bool = false, use_space: bool = false) const 🔗

Returns the current date and time as an ISO 8601 date and time string (YYYY-MM-DDTHH:MM:SS).

The returned values are in the system's local time when `utc` is `false`, otherwise they are in UTC.

If `use_space` is `true`, the date and time bits are separated by an empty space character instead of the letter T.

String **get_datetime_string_from_unix_time**(unix_time_val: int, use_space: bool = false) const

Converts the given Unix timestamp to an ISO 8601 date and time string (YYYY-MM-DDTHH:MM:SS).

If `use_space` is `true`, the date and time bits are separated by an empty space character instead of the letter T.

String **get_offset_string_from_offset_minutes**(offset_minutes: int) const

Converts the given timezone offset in minutes to a timezone offset string. For example, -480 returns "-08:00", 345 returns "+05:45", and 0 returns "+00:00".

---

int **get_ticks_msec**() const

Returns the amount of time passed in milliseconds since the engine started.

Will always be positive or 0 and uses a 64-bit value (it will wrap after roughly 500 million years).

---

int **get_ticks_usec**() const

Returns the amount of time passed in microseconds since the engine started.

Will always be positive or 0 and uses a 64-bit value (it will wrap after roughly half a million years).

Dictionary **get_time_dict_from_system**(utc: bool = false) const 🔗

Returns the current time as a dictionary of keys: `hour`, `minute`, and `second`.

The returned values are in the system's local time when `utc` is `false`, otherwise they are in UTC.

Dictionary **get_time_dict_from_unix_time**(unix_time_val: int) const 🔗

Converts the given time to a dictionary of keys: `hour`, `minute`, and `second`.

String **get_time_string_from_system**(utc: bool = false) const 🔗

Returns the current time as an ISO 8601 time string (HH:MM:SS).

The returned values are in the system's local time when `utc` is `false`, otherwise they are in UTC.

String **get_time_string_from_unix_time**(unix_time_val: int) const 🔗

Converts the given Unix timestamp to an ISO 8601 time string (HH:MM:SS).

Dictionary **get_time_zone_from_system**() const

Returns the current time zone as a dictionary of keys: `bias` and `name`.

- `bias` is the offset from UTC in minutes, since not all time zones are multiples of an hour from UTC.
- `name` is the localized name of the time zone, according to the OS locale settings of the current user.

---

int **get_unix_time_from_datetime_dict**(datetime: Dictionary) const

Converts a dictionary of time values to a Unix timestamp.

The given dictionary can be populated with the following keys: `year`, `month`, `day`, `hour`, `minute`, and `second`. Any other entries (including `dst`) are ignored.

If the dictionary is empty, `0` is returned. If some keys are omitted, they default to the equivalent values for the Unix epoch timestamp 0 (1970-01-01 at 00:00:00).

You can pass the output from get_datetime_dict_from_unix_time directly into this function and get the same as what was put in.

Note: Unix timestamps are often in UTC. This method does not do any timezone conversion, so the timestamp will be in the same timezone as the given datetime dictionary.

---

int **get_unix_time_from_datetime_string**(datetime: String) const

Converts the given ISO 8601 date and/or time string to a Unix timestamp. The string can contain a date only, a time only, or both.

Note: Unix timestamps are often in UTC. This method does not do any timezone conversion, so the timestamp will be in the same timezone as the given datetime string.

Note: Any decimal fraction in the time string will be ignored silently.

---

float **get_unix_time_from_system**() const 🔗

Returns the current Unix timestamp in seconds based on the system time in UTC. This method is implemented by the operating system and always returns the time in UTC. The Unix timestamp is the number of seconds passed since 1970-01-01 at 00:00:00, the Unix epoch [https://en.wikipedia.org/wiki/Unix_time].

Note: Unlike other methods that use integer timestamps, this method returns the timestamp as a float for sub-second precision.

TLSOptions

Inherits: RefCounted **<** Object

TLS configuration for clients and servers.

Description

TLSOptions abstracts the configuration options for the StreamPeerTLS and PacketPeerDTLS classes.

Objects of this class cannot be instantiated directly, and one of the static methods client, client_unsafe, or server should be used instead.

GDScript

```
# Create a TLS client configuration which uses our custom trusted CA chain.
var client_trusted_cas = load("res://my_trusted_cas.crt")
var client_tls_options = TLSOptions.client(client_trusted_cas)

# Create a TLS server configuration.
var server_certs = load("res://my_server_cas.crt")
var server_key = load("res://my_server_key.key")
var server_tls_options = TLSOptions.server(server_key, server_certs)
```

Methods

TLSOptions	client(trusted_chain: X509Certificate = null, common_name_override: String = "") static

TLSOptions	client_unsafe(trusted_chain: X509Certificate = null) static
String	get_common_name_override() const
X509Certificate	get_own_certificate() const
CryptoKey	get_private_key() const
X509Certificate	get_trusted_ca_chain() const
bool	is_server() const
bool	is_unsafe_client() const
TLSOptions	server(key: CryptoKey, certificate: X509Certificate) static

Method Descriptions

TLSOptions **client**(trusted_chain: X509Certificate = null, common_name_override: String = "") static 🔗

Creates a TLS client configuration which validates certificates and their common names (fully qualified domain names).

You can specify a custom `trusted_chain` of certification authorities (the default CA list will be used if `null`), and optionally provide a `common_name_override` if you expect the certificate to have a common name other than the server FQDN.

Note: On the Web platform, TLS verification is always enforced against the CA list of the web browser. This is considered a security feature.

TLSOptions **client_unsafe**(trusted_chain: X509Certificate = null) static

Creates an **unsafe** TLS client configuration where certificate validation is optional. You can optionally provide a valid `trusted_chain`, but the common name of the certificates will never be checked. Using this configuration for purposes other than testing **is not recommended**.

Note: On the Web platform, TLS verification is always enforced against the CA list of the web browser. This is considered a security feature.

String **get_common_name_override**() const

Returns the common name (domain name) override specified when creating with client.

X509Certificate **get_own_certificate**() const

Returns the X509Certificate specified when creating with server.

CryptoKey **get_private_key**() const

Returns the CryptoKey specified when creating with server.

X509Certificate **get_trusted_ca_chain**() const

Returns the CA X509Certificate chain specified when creating with client or client_unsafe.

---

bool **is_server**() const

Returns `true` if created with server, `false` otherwise.

---

bool **is_unsafe_client**() const

Returns `true` if created with client_unsafe, `false` otherwise.

TLSOptions **server**(key: CryptoKey, certificate: X509Certificate) static

Creates a TLS server configuration using the provided `key` and `certificate`.

Note: The `certificate` should include the full certificate chain up to the signing CA (certificates file can be concatenated using a general purpose text editor).

TranslationServer

Inherits: Object

The server responsible for language translations.

Description

The server that manages all language translations. Translations can be added to or removed from it.

Tutorials

- Internationalizing games
- Locales

Properties

bool	pseudolocalization_enabled	`false`

Methods

void	add_translation(translation: Translation)
void	clear()

int	compare_locales(locale_a: String, locale_b: String) const
PackedStringArray	get_all_countries() const
PackedStringArray	get_all_languages() const
PackedStringArray	get_all_scripts() const
String	get_country_name(country: String) const
String	get_language_name(language: String) const
PackedStringArray	get_loaded_locales() const
String	get_locale() const
String	get_locale_name(locale: String) const
String	get_script_name(script: String) const
String	get_tool_locale()
Translation	get_translation_object(locale: String)

StringName	pseudolocalize(message: StringName) const
void	reload_pseudolocalization()
void	remove_translation(translation: Translation)
void	set_locale(locale: String)
String	standardize_locale(locale: String) const
StringName	translate(message: StringName, context: StringName = &"") const
StringName	translate_plural(message: StringName, plural_message: StringName, n: int, context: StringName = &"") const

---

Property Descriptions

bool **pseudolocalization_enabled** = `false` 🔗

- void **set_pseudolocalization_enabled**(value: bool)
- bool **is_pseudolocalization_enabled**()

If `true`, enables the use of pseudolocalization. See ProjectSettings.internationalization/pseudolocalization/use_pseudolocalization for details.

Method Descriptions

void **add_translation**(translation: Translation)

Adds a Translation resource.

---

void **clear**()

Clears the server from all translations.

---

int **compare_locales**(locale_a: String, locale_b: String) const

Compares two locales and returns a similarity score between `0` (no match) and `10` (full match).

PackedStringArray **get_all_countries**() const

Returns an array of known country codes.

PackedStringArray **get_all_languages**() const

Returns array of known language codes.

PackedStringArray **get_all_scripts**() const

Returns an array of known script codes.

String **get_country_name**(country: String) const

Returns a readable country name for the `country` code.

String **get_language_name**(language: String) const 🔗

Returns a readable language name for the `language` code.

PackedStringArray **get_loaded_locales**() const 🔗

Returns an array of all loaded locales of the project.

String **get_locale**() const 🔗

Returns the current locale of the project.

See also OS.get_locale and OS.get_locale_language to query the locale of the user system.

String **get_locale_name**(locale: String) const 🔗

Returns a locale's language and its variant (e.g. `"en_US"` would return `"English (United States)"`).

String **get_script_name**(script: String) const 🔗

Returns a readable script name for the `script` code.

String **get_tool_locale**() 🔗

Returns the current locale of the editor.

Note: When called from an exported project returns the same value as get_locale.

Translation **get_translation_object**(locale: String)

Returns the Translation instance based on the `locale` passed in.

It will return `null` if there is no Translation instance that matches the `locale`.

StringName **pseudolocalize**(message: StringName) const

Returns the pseudolocalized string based on the `message` passed in.

---

void **reload_pseudolocalization**()

Reparses the pseudolocalization options and reloads the translation.

---

void **remove_translation**(translation: Translation)

Removes the given translation from the server.

---

void **set_locale**(locale: String)

Sets the locale of the project. The `locale` string will be standardized to match known locales (e.g. `en-US` would be matched to `en_US`).

If translations have been loaded beforehand for the new locale, they will be applied.

String **standardize_locale**(locale: String) const

Returns a `locale` string standardized to match known locales (e.g. `en-US` would be matched to `en_US`).

StringName **translate**(message: StringName, context: StringName = &"") const 🔗

Returns the current locale's translation for the given message (key) and context.

StringName **translate_plural**(message: StringName, plural_message: StringName, n: int, context: StringName = &"") const 🔗

Returns the current locale's translation for the given message (key), plural message and context.

The number `n` is the number or quantity of the plural object. It will be used to guide the translation system to fetch the correct plural form for the selected language.

TreeItem

Inherits: Object

An internal control for a single item inside Tree.

Description

A single item of a Tree control. It can contain other **TreeItem**s as children, which allows it to create a hierarchy. It can also contain text and buttons. **TreeItem** is not a Node, it is internal to the Tree.

To create a **TreeItem**, use Tree.create_item or create_child. To remove a **TreeItem**, use Object.free.

Note: The ID values used for buttons are 32-bit, unlike int which is always 64-bit. They go from `-2147483648` to `2147483647`.

Properties

bool	collapsed
int	custom_minimum_height
bool	disable_folding
bool	visible

Methods

void	add_button(column: int, button: Texture2D, id: int = -1, disabled: bool = false, tooltip_text: String = "")
void	add_child(child: TreeItem)
void	call_recursive(method: StringName, ...) vararg
void	clear_custom_bg_color(column: int)
void	clear_custom_color(column: int)
TreeItem	create_child(index: int = -1)
void	deselect(column: int)
void	erase_button(column: int, button_index: int)
AutowrapMode	get_autowrap_mode(column: int) const
Texture2D	get_button(column: int, button_index: int) const
int	get_button_by_id(column: int, id: int) const
Color	get_button_color(column: int, id: int) const
int	get_button_count(column: int) const

int	get_button_id(column: int, button_index: int) const
String	get_button_tooltip_text(column: int, button_index: int) const
TreeCellMode	get_cell_mode(column: int) const
TreeItem	get_child(index: int)
int	get_child_count()
Array[TreeItem]	get_children()
Color	get_custom_bg_color(column: int) const
Color	get_custom_color(column: int) const
Callable	get_custom_draw_callback(column: int) const
Font	get_custom_font(column: int) const
int	get_custom_font_size(column: int) const
bool	get_expand_right(column: int) const
TreeItem	get_first_child() const

Texture2D	get_icon(column: int) const
int	get_icon_max_width(column: int) const
Color	get_icon_modulate(column: int) const
Rect2	get_icon_region(column: int) const
int	get_index()
String	get_language(column: int) const
Variant	get_metadata(column: int) const
TreeItem	get_next() const
TreeItem	get_next_in_tree(wrap: bool = false)
TreeItem	get_next_visible(wrap: bool = false)
TreeItem	get_parent() const
TreeItem	get_prev()
TreeItem	get_prev_in_tree(wrap: bool = false)
TreeItem	get_prev_visible(wrap: bool = false)

float	get_range(column: int) const
Dictionary	get_range_config(column: int)
StructuredTextParser	get_structured_text_bidi_override(column: int) const
Array	get_structured_text_bidi_override_options(column: int) const
String	get_suffix(column: int) const
String	get_text(column: int) const
HorizontalAlignment	get_text_alignment(column: int) const
TextDirection	get_text_direction(column: int) const
OverrunBehavior	get_text_overrun_behavior(column: int) const
String	get_tooltip_text(column: int) const
Tree	get_tree() const
bool	is_any_collapsed(only_visible: bool = false)

bool	is_button_disabled(column: int, button_index: int) const
bool	is_checked(column: int) const
bool	is_custom_set_as_button(column: int) const
bool	is_edit_multiline(column: int) const
bool	is_editable(column: int)
bool	is_indeterminate(column: int) const
bool	is_selectable(column: int) const
bool	is_selected(column: int)
bool	is_visible_in_tree() const
void	move_after(item: TreeItem)
void	move_before(item: TreeItem)
void	propagate_check(column: int, emit_signal: bool = true)
void	remove_child(child: TreeItem)

void	select(column: int)
void	set_autowrap_mode(column: int, autowrap_mode: AutowrapMode)
void	set_button(column: int, button_index: int, button: Texture2D)
void	set_button_color(column: int, button_index: int, color: Color)
void	set_button_disabled(column: int, button_index: int, disabled: bool)
void	set_button_tooltip_text(column: int, button_index: int, tooltip: String)
void	set_cell_mode(column: int, mode: TreeCellMode)
void	set_checked(column: int, checked: bool)
void	set_collapsed_recursive(enable: bool)
void	set_custom_as_button(column: int, enable: bool)
void	set_custom_bg_color(column: int, color: Color, just_outline: bool = false)

void	set_custom_color(column: int, color: Color)
void	set_custom_draw(column: int, object: Object, callback: StringName)
void	set_custom_draw_callback(column: int, callback: Callable)
void	set_custom_font(column: int, font: Font)
void	set_custom_font_size(column: int, font_size: int)
void	set_edit_multiline(column: int, multiline: bool)
void	set_editable(column: int, enabled: bool)
void	set_expand_right(column: int, enable: bool)
void	set_icon(column: int, texture: Texture2D)
void	set_icon_max_width(column: int, width: int)
void	set_icon_modulate(column: int, modulate: Color)
void	set_icon_region(column: int, region: Rect2)
void	set_indeterminate(column: int, indeterminate: bool)

void	set_language(column: int, language: String)
void	set_metadata(column: int, meta: Variant)
void	set_range(column: int, value: float)
void	set_range_config(column: int, min: float, max: float, step: float, expr: bool = false)
void	set_selectable(column: int, selectable: bool)
void	set_structured_text_bidi_override(column: int, parser: StructuredTextParser)
void	set_structured_text_bidi_override_options(column: int, args: Array)
void	set_suffix(column: int, text: String)
void	set_text(column: int, text: String)
void	set_text_alignment(column: int, text_alignment: HorizontalAlignment)
void	set_text_direction(column: int, direction: TextDirection)

void	set_text_overrun_behavior(column: int, overrun_behavior: OverrunBehavior)
void	set_tooltip_text(column: int, tooltip: String)
void	uncollapse_tree()

Enumerations

enum **TreeCellMode**:

TreeCellMode **CELL_MODE_STRING** = `0`

Cell shows a string label. When editable, the text can be edited using a LineEdit, or a TextEdit popup if set_edit_multiline is used.

TreeCellMode **CELL_MODE_CHECK** = `1`

Cell shows a checkbox, optionally with text. The checkbox can be pressed, released, or indeterminate (via set_indeterminate). The checkbox can't be clicked unless the cell is editable.

TreeCellMode **CELL_MODE_RANGE** = `2`

Cell shows a numeric range. When editable, it can be edited using a range slider. Use set_range to set the value and set_range_config to configure the range.

This cell can also be used in a text dropdown mode when you assign a text with set_text. Separate options with a comma, e.g. `"Option1,Option2,Option3"`.

TreeCellMode **CELL_MODE_ICON** = `3`

Cell shows an icon. It can't be edited nor display text.

TreeCellMode **CELL_MODE_CUSTOM** = `4`

Cell shows as a clickable button. It will display an arrow similar to OptionButton, but doesn't feature a dropdown (for that you can use CELL_MODE_RANGE). Clicking the button emits the Tree.item_edited signal. The button is flat by default, you can use set_custom_as_button to display it with a StyleBox.

This mode also supports custom drawing using set_custom_draw_callback.

Property Descriptions

bool **collapsed** 🔗

- void **set_collapsed**(value: bool)
- bool **is_collapsed**()

If `true`, the TreeItem is collapsed.

---

int **custom_minimum_height** 🔗

- void **set_custom_minimum_height**(value: int)
- int **get_custom_minimum_height**()

The custom minimum height.

---

bool **disable_folding** 🔗

- void **set_disable_folding**(value: bool)
- bool **is_folding_disabled**()

If `true`, folding is disabled for this TreeItem.

---

bool **visible** 🔗

- void **set_visible**(value: bool)
- bool **is_visible**()

If `true`, the **TreeItem** is visible (default).

Note that if a **TreeItem** is set to not be visible, none of its children will be visible either.

Method Descriptions

void **add_button**(column: int, button: Texture2D, id: int = -1, disabled: bool = false, tooltip_text: String = "") 🔗

Adds a button with Texture2D `button` at column `column`. The `id` is used to identify the button in the according Tree.button_clicked signal and can be different from the buttons index. If not specified, the next available index is used, which may be retrieved by calling get_button_count immediately before this method. Optionally, the button can be `disabled` and have a `tooltip_text`.

---

void **add_child**(child: TreeItem) 🔗

Adds a previously unparented **TreeItem** as a direct child of this one. The `child` item must not be a part of any Tree or parented to any **TreeItem**. See also remove_child.

---

void **call_recursive**(method: StringName, ...) vararg 🔗

Calls the `method` on the actual TreeItem and its children recursively. Pass parameters as a comma separated list.

---

void **clear_custom_bg_color**(column: int) 🔗

Resets the background color for the given column to default.

---

void **clear_custom_color**(column: int)

Resets the color for the given column to default.

TreeItem **create_child**(index: int = -1)

Creates an item and adds it as a child.

The new item will be inserted as position `index` (the default value `-1` means the last position), or it will be the last child if `index` is higher than the child count.

---

void **deselect**(column: int)

Deselects the given column.

---

void **erase_button**(column: int, button_index: int)

Removes the button at index `button_index` in column `column`.

AutowrapMode **get_autowrap_mode**(column: int) const

Returns the text autowrap mode in the given `column`. By default it is TextServer.AUTOWRAP_OFF.

Texture2D **get_button**(column: int, button_index: int) const

Returns the Texture2D of the button at index `button_index` in column `column`.

---

int **get_button_by_id**(column: int, id: int) const

Returns the button index if there is a button with ID `id` in column `column`, otherwise returns -1.

Color **get_button_color**(column: int, id: int) const

Returns the color of the button with ID `id` in column `column`. If the specified button does not exist, returns Color.BLACK.

---

int **get_button_count**(column: int) const

Returns the number of buttons in column `column`.

---

int **get_button_id**(column: int, button_index: int) const

Returns the ID for the button at index `button_index` in column `column`.

String **get_button_tooltip_text**(column: int, button_index: int) const

Returns the tooltip text for the button at index `button_index` in column `column`.

TreeCellMode **get_cell_mode**(column: int) const

Returns the column's cell mode.

TreeItem **get_child**(index: int)

Returns a child item by its `index` (see get_child_count). This method is often used for iterating all children of an item.

Negative indices access the children from the last one.

int **get_child_count**() 🔗

Returns the number of child items.

Array[TreeItem] **get_children**() 🔗

Returns an array of references to the item's children.

Color **get_custom_bg_color**(column: int) const 🔗

Returns the custom background color of column `column`.

Color **get_custom_color**(column: int) const 🔗

Returns the custom color of column `column`.

Callable **get_custom_draw_callback**(column: int) const 🔗

Returns the custom callback of column `column`.

Font **get_custom_font**(column: int) const 🔗

Returns custom font used to draw text in the column `column`.

int **get_custom_font_size**(column: int) const 🔗

Returns custom font size used to draw text in the column `column`.

bool **get_expand_right**(column: int) const 🔗

Returns `true` if `expand_right` is set.

TreeItem **get_first_child**() const

Returns the TreeItem's first child.

Texture2D **get_icon**(column: int) const

Returns the given column's icon Texture2D. Error if no icon is set.

int **get_icon_max_width**(column: int) const

Returns the maximum allowed width of the icon in the given `column`.

Color **get_icon_modulate**(column: int) const

Returns the Color modulating the column's icon.

Rect2 **get_icon_region**(column: int) const

Returns the icon Texture2D region as Rect2.

int **get_index**()

Returns the node's order in the tree. For example, if called on the first child item the position is `0`.

String **get_language**(column: int) const

Returns item's text language code.

Variant **get_metadata**(column: int) const

Returns the metadata value that was set for the given column using set_metadata.

TreeItem **get_next**() const

Returns the next sibling TreeItem in the tree or a null object if there is none.

TreeItem **get_next_in_tree**(wrap: bool = false)

Returns the next TreeItem in the tree (in the context of a depth-first search) or a `null` object if there is none.

If `wrap` is enabled, the method will wrap around to the first element in the tree when called on the last element, otherwise it returns `null`.

TreeItem **get_next_visible**(wrap: bool = false)

Returns the next visible TreeItem in the tree (in the context of a depth-first search) or a `null` object if there is none.

If `wrap` is enabled, the method will wrap around to the first visible element in the tree when called on the last visible element, otherwise it returns `null`.

TreeItem **get_parent**() const

Returns the parent TreeItem or a null object if there is none.

TreeItem **get_prev**()

Returns the previous sibling TreeItem in the tree or a null object if there is none.

TreeItem **get_prev_in_tree**(wrap: bool = false)

Returns the previous TreeItem in the tree (in the context of a depth-first search) or a `null` object if there is none.

If `wrap` is enabled, the method will wrap around to the last element in the tree when called on the first visible element, otherwise it returns `null`.

TreeItem **get_prev_visible**(wrap: bool = false)

Returns the previous visible sibling TreeItem in the tree (in the context of a depth-first search) or a `null` object if there is none.

If `wrap` is enabled, the method will wrap around to the last visible element in the tree when called on the first visible element, otherwise it returns `null`.

---

float **get_range**(column: int) const

Returns the value of a CELL_MODE_RANGE column.

Dictionary **get_range_config**(column: int)

Returns a dictionary containing the range parameters for a given column. The keys are "min", "max", "step", and "expr".

StructuredTextParser **get_structured_text_bidi_override**(column: int) const

Returns the BiDi algorithm override set for this cell.

Array **get_structured_text_bidi_override_options**(column: int) const

Returns the additional BiDi options set for this cell.

String **get_suffix**(column: int) const

Gets the suffix string shown after the column value.

String **get_text**(column: int) const

Returns the given column's text.

HorizontalAlignment **get_text_alignment**(column: int) const

Returns the given column's text alignment.

TextDirection **get_text_direction**(column: int) const

Returns item's text base writing direction.

OverrunBehavior **get_text_overrun_behavior**(column: int) const

Returns the clipping behavior when the text exceeds the item's bounding rectangle in the given `column`. By default it is TextServer.OVERRUN_TRIM_ELLIPSIS.

String **get_tooltip_text**(column: int) const

Returns the given column's tooltip text.

Tree **get_tree**() const

Returns the Tree that owns this TreeItem.

---

bool **is_any_collapsed**(only_visible: bool = false)

Returns `true` if this **TreeItem**, or any of its descendants, is collapsed.

If `only_visible` is `true` it ignores non-visible **TreeItem**s.

---

bool **is_button_disabled**(column: int, button_index: int) const

Returns `true` if the button at index `button_index` for the given `column` is disabled.

---

bool **is_checked**(column: int) const

Returns `true` if the given `column` is checked.

---

bool **is_custom_set_as_button**(column: int) const

Returns `true` if the cell was made into a button with set_custom_as_button.

---

bool **is_edit_multiline**(column: int) const

Returns `true` if the given `column` is multiline editable.

---

bool **is_editable**(column: int)

Returns `true` if the given `column` is editable.

---

bool **is_indeterminate**(column: int) const

Returns `true` if the given `column` is indeterminate.

---

bool **is_selectable**(column: int) const

Returns `true` if the given `column` is selectable.

---

bool **is_selected**(column: int)

Returns `true` if the given `column` is selected.

---

bool **is_visible_in_tree**() const

Returns `true` if visible is `true` and all its ancestors are also visible.

---

void **move_after**(item: TreeItem)

Moves this TreeItem right after the given `item`.

Note: You can't move to the root or move the root.

---

void **move_before**(item: TreeItem)

Moves this TreeItem right before the given `item`.

Note: You can't move to the root or move the root.

---

void **propagate_check**(column: int, emit_signal: bool = true)

Propagates this item's checked status to its children and parents for the given `column`. It is possible to process the items affected by this method call by connecting to Tree.check_propagated_to_item. The order that the items affected will be processed is as follows: the item invoking this method, children of that item, and finally parents of that item. If `emit_signal` is `false`, then Tree.check_propagated_to_item will not be emitted.

---

void **remove_child**(child: TreeItem)

Removes the given child **TreeItem** and all its children from the Tree. Note that it doesn't free the item from memory, so it can be reused later (see add_child). To completely remove a **TreeItem** use Object.free.

Note: If you want to move a child from one Tree to another, then instead of removing and adding it manually you can use move_before or move_after.

---

void **select**(column: int)

Selects the given `column`.

---

void **set_autowrap_mode**(column: int, autowrap_mode: AutowrapMode)

Sets the autowrap mode in the given `column`. If set to something other than TextServer.AUTOWRAP_OFF, the text gets wrapped inside the cell's bounding rectangle.

---

void **set_button**(column: int, button_index: int, button: Texture2D)

Sets the given column's button Texture2D at index `button_index` to `button`.

---

void **set_button_color**(column: int, button_index: int, color: Color)

Sets the given column's button color at index `button_index` to `color`.

---

void **set_button_disabled**(column: int, button_index: int, disabled: bool)

If `true`, disables the button at index `button_index` in the given `column`.

---

void **set_button_tooltip_text**(column: int, button_index: int, tooltip: String)

Sets the tooltip text for the button at index `button_index` in the given `column`.

---

void **set_cell_mode**(column: int, mode: TreeCellMode)

Sets the given column's cell mode to `mode`. This determines how the cell is displayed and edited. See TreeCellMode constants for details.

---

void **set_checked**(column: int, checked: bool)

If `checked` is `true`, the given `column` is checked. Clears column's indeterminate status.

---

void **set_collapsed_recursive**(enable: bool)

Collapses or uncollapses this **TreeItem** and all the descendants of this item.

---

void **set_custom_as_button**(column: int, enable: bool)

Makes a cell with CELL_MODE_CUSTOM display as a non-flat button with a StyleBox.

---

void **set_custom_bg_color**(column: int, color: Color, just_outline: bool = false)

Sets the given column's custom background color and whether to just use it as an outline.

---

void **set_custom_color**(column: int, color: Color)

Sets the given column's custom color.

---

void **set_custom_draw**(column: int, object: Object, callback: StringName)

Deprecated: Use set_custom_draw_callback instead.

Sets the given column's custom draw callback to the `callback` method on `object`.

The method named `callback` should accept two arguments: the **TreeItem** that is drawn and its position and size as a Rect2.

---

void **set_custom_draw_callback**(column: int, callback: Callable)

Sets the given column's custom draw callback. Use an empty Callable (`Callable()`) to clear the custom callback. The cell has to be in CELL_MODE_CUSTOM to use this feature.

The `callback` should accept two arguments: the **TreeItem** that is drawn and its position and size as a Rect2.

---

void **set_custom_font**(column: int, font: Font)

Sets custom font used to draw text in the given `column`.

---

void **set_custom_font_size**(column: int, font_size: int)

Sets custom font size used to draw text in the given `column`.

---

void **set_edit_multiline**(column: int, multiline: bool)

If `multiline` is `true`, the given `column` is multiline editable.

Note: This option only affects the type of control (LineEdit or TextEdit) that appears when editing the column. You can set multiline values with set_text even if the column is not multiline editable.

---

void **set_editable**(column: int, enabled: bool)

If `enabled` is `true`, the given `column` is editable.

---

void **set_expand_right**(column: int, enable: bool)

If `enable` is `true`, the given `column` is expanded to the right.

---

void **set_icon**(column: int, texture: Texture2D)

Sets the given cell's icon Texture2D. The cell has to be in CELL_MODE_ICON mode.

---

void **set_icon_max_width**(column: int, width: int)

Sets the maximum allowed width of the icon in the given `column`. This limit is applied on top of the default size of the icon and on top of Tree.icon_max_width. The height is adjusted according to the icon's ratio.

---

void **set_icon_modulate**(column: int, modulate: Color)

Modulates the given column's icon with `modulate`.

---

void **set_icon_region**(column: int, region: Rect2)

Sets the given column's icon's texture region.

---

void **set_indeterminate**(column: int, indeterminate: bool)

If `indeterminate` is `true`, the given `column` is marked indeterminate.

Note: If set `true` from `false`, then column is cleared of checked status.

---

void **set_language**(column: int, language: String)

Sets language code of item's text used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

---

void **set_metadata**(column: int, meta: Variant)

Sets the metadata value for the given column, which can be retrieved later using get_metadata. This can be used, for example, to store a reference to the original data.

---

void **set_range**(column: int, value: float)

Sets the value of a CELL_MODE_RANGE column.

---

void **set_range_config**(column: int, min: float, max: float, step: float, expr: bool = false)

Sets the range of accepted values for a column. The column must be in the CELL_MODE_RANGE mode.

If `expr` is `true`, the edit mode slider will use an exponential scale as with Range.exp_edit.

---

void **set_selectable**(column: int, selectable: bool)

If `selectable` is `true`, the given `column` is selectable.

void **set_structured_text_bidi_override**(column: int, parser: StructuredTextParser)

Set BiDi algorithm override for the structured text. Has effect for cells that display text.

void **set_structured_text_bidi_override_options**(column: int, args: Array)

Set additional options for BiDi override. Has effect for cells that display text.

void **set_suffix**(column: int, text: String)

Sets a string to be shown after a column's value (for example, a unit abbreviation).

void **set_text**(column: int, text: String)

Sets the given column's text value.

void **set_text_alignment**(column: int, text_alignment: HorizontalAlignment)

Sets the given column's text alignment. See HorizontalAlignment for possible values.

void **set_text_direction**(column: int, direction: TextDirection)

Sets item's text base writing direction.

void **set_text_overrun_behavior**(column: int, overrun_behavior: OverrunBehavior)

Sets the clipping behavior when the text exceeds the item's bounding rectangle in the given `column`.

---

void **set_tooltip_text**(column: int, tooltip: String)

Sets the given column's tooltip text.

---

void **uncollapse_tree**()

Uncollapses all **TreeItem**s necessary to reveal this **TreeItem**, i.e. all ancestor **TreeItem**s.

TriangleMesh

Inherits: RefCounted **<** Object

Internal mesh type.

Description

Mesh type used internally for collision calculations.

Tween

Inherits: RefCounted **<** Object

Lightweight object used for general-purpose animation via script, using Tweeners.

Description

Tweens are mostly useful for animations requiring a numerical property to be interpolated over a range of values. The name *tween* comes from *in-betweening*, an animation technique where you specify *keyframes* and the computer interpolates the frames that appear between them. Animating something with a **Tween** is called tweening.

Tween is more suited than AnimationPlayer for animations where you don't know the final values in advance. For example, interpolating a dynamically-chosen camera zoom value is best done with a **Tween**; it would be difficult to do the same thing with an AnimationPlayer node. Tweens are also more light-weight than AnimationPlayer, so they are very much suited for simple animations or general tasks that don't require visual tweaking provided by the editor. They can be used in a "fire-and-forget" manner for some logic that normally would be done by code. You can e.g. make something shoot periodically by using a looped CallbackTweener with a delay.

A **Tween** can be created by using either SceneTree.create_tween or Node.create_tween. **Tween**s created manually (i.e. by using `Tween.new()`) are invalid and can't be used for tweening values.

A tween animation is created by adding Tweeners to the **Tween** object, using tween_property, tween_interval, tween_callback or tween_method:

GDScript

```
var tween = get_tree().create_tween()
tween.tween_property($Sprite, "modulate", Color.RED, 1)
tween.tween_property($Sprite, "scale", Vector2(), 1)
tween.tween_callback($Sprite.queue_free)
```

C#

```
Tween tween = GetTree().CreateTween();
tween.TweenProperty(GetNode("Sprite"), "modulate", Colors.Red,
1.0f);
tween.TweenProperty(GetNode("Sprite"), "scale", Vector2.Zero,
1.0f);
tween.TweenCallback(Callable.From(GetNode("Sprite").QueueFree))
;
```

This sequence will make the `$Sprite` node turn red, then shrink, before finally calling Node.queue_free to free the sprite. Tweeners are executed one after another by default. This behavior can be changed using parallel and set_parallel.

When a Tweener is created with one of the `tween_*` methods, a chained method call can be used to tweak the properties of this Tweener. For example, if you want to set a different transition type in the above example, you can use set_trans:

GDScript

```
var tween = get_tree().create_tween()
tween.tween_property($Sprite, "modulate", Color.RED,
1).set_trans(Tween.TRANS_SINE)
tween.tween_property($Sprite, "scale", Vector2(),
1).set_trans(Tween.TRANS_BOUNCE)
tween.tween_callback($Sprite.queue_free)
```

C#

```
Tween tween = GetTree().CreateTween();
tween.TweenProperty(GetNode("Sprite"), "modulate", Colors.Red, 1.0f).SetTrans(Tween.TransitionType.Sine);
tween.TweenProperty(GetNode("Sprite"), "scale", Vector2.Zero, 1.0f).SetTrans(Tween.TransitionType.Bounce);
tween.TweenCallback(Callable.From(GetNode("Sprite").QueueFree));
```

Most of the **Tween** methods can be chained this way too. In the following example the **Tween** is bound to the running script's node and a default transition is set for its Tweeners:

GDScript

```
var tween = get_tree().create_tween().bind_node(self).set_trans(Tween.TRANS_ELASTIC)
tween.tween_property($Sprite, "modulate", Color.RED, 1)
tween.tween_property($Sprite, "scale", Vector2(), 1)
tween.tween_callback($Sprite.queue_free)
```

C#

```
var tween = GetTree().CreateTween().BindNode(this).SetTrans(Tween.TransitionType.Elastic);
tween.TweenProperty(GetNode("Sprite"), "modulate", Colors.Red, 1.0f);
tween.TweenProperty(GetNode("Sprite"), "scale", Vector2.Zero, 1.0f);
tween.TweenCallback(Callable.From(GetNode("Sprite").QueueFree));
```

Another interesting use for **Tweens** is animating arbitrary sets of objects:

GDScript

```
var tween = create_tween()
for sprite in get_children():
    tween.tween_property(sprite, "position", Vector2(0, 0), 1)
```

C#

```
Tween tween = CreateTween();
foreach (Node sprite in GetChildren())
    tween.TweenProperty(sprite, "position", Vector2.Zero,
1.0f);
```

In the example above, all children of a node are moved one after another to position (0, 0).

You should avoid using more than one **Tween** per object's property. If two or more tweens animate one property at the same time, the last one created will take priority and assign the final value. If you want to interrupt and restart an animation, consider assigning the **Tween** to a variable:

GDScript

```
var tween
func animate():
    if tween:
        tween.kill() # Abort the previous animation.
    tween = create_tween()
```

C#

```
private Tween _tween;

public void Animate()
{
    if (_tween != null)
        _tween.Kill(); // Abort the previous animation
    _tween = CreateTween();
}
```

Some Tweeners use transitions and eases. The first accepts a TransitionType constant, and refers to the way the timing of the animation is handled (see easings.net [https://easings.net/] for some examples). The second accepts an EaseType constant, and controls where the `trans_type` is applied to the interpolation (in the beginning, the end, or both). If you don't know which transition and easing to pick, you can try different TransitionType constants with EASE_IN_OUT, and use the one that looks best.

Tween easing and transition types cheatsheet [https://raw.githubusercontent.com/godotengine/godot-docs/master/img/tween_cheatsheet.webp]

Note: Tweens are not designed to be re-used and trying to do so results in an undefined behavior. Create a new Tween for each animation and every time you replay an animation from start. Keep in mind that Tweens start immediately, so only create a Tween when you want to start animating.

Note: The tween is processed after all of the nodes in the current frame, i.e. node's Node._process method would be called before the tween (or Node._physics_process depending on the value passed to set_process_mode).

Methods

Tween	bind_node(node: Node)
Tween	chain()
bool	custom_step(delta: float)
int	get_loops_left() const

float	get_total_elapsed_time() const
Variant	interpolate_value(initial_value: Variant, delta_value: Variant, elapsed_time: float, duration: float, trans_type: TransitionType, ease_type: EaseType) static
bool	is_running()
bool	is_valid()
void	kill()
Tween	parallel()
void	pause()
void	play()
Tween	set_ease(ease: EaseType)
Tween	set_loops(loops: int = 0)
Tween	set_parallel(parallel: bool = true)
Tween	set_pause_mode(mode: TweenPauseMode)

Tween	set_process_mode(mode: TweenProcessMode)
Tween	set_speed_scale(speed: float)
Tween	set_trans(trans: TransitionType)
void	stop()
CallbackTweener	tween_callback(callback: Callable)
IntervalTweener	tween_interval(time: float)
MethodTweener	tween_method(method: Callable, from: Variant, to: Variant, duration: float)
PropertyTweener	tween_property(object: Object, property: NodePath, final_val: Variant, duration: float)

Signals

finished()

Emitted when the **Tween** has finished all tweening. Never emitted when the **Tween** is set to infinite looping (see set_loops).

loop_finished(loop_count: int)

Emitted when a full loop is complete (see set_loops), providing the loop index. This signal is not emitted after the final loop, use finished instead for this case.

step_finished(idx: int)

Emitted when one step of the **Tween** is complete, providing the step index. One step is either a single Tweener or a group of Tweeners running in parallel.

Enumerations

enum **TweenProcessMode**:

TweenProcessMode **TWEEN_PROCESS_PHYSICS** = `0`

The **Tween** updates after each physics frame (see Node._physics_process).

TweenProcessMode **TWEEN_PROCESS_IDLE** = `1`

The **Tween** updates after each process frame (see Node._process).

enum **TweenPauseMode**:

TweenPauseMode **TWEEN_PAUSE_BOUND** = `0`

If the **Tween** has a bound node, it will process when that node can process (see Node.process_mode). Otherwise it's the same as TWEEN_PAUSE_STOP.

TweenPauseMode **TWEEN_PAUSE_STOP** = `1`

If SceneTree is paused, the **Tween** will also pause.

TweenPauseMode **TWEEN_PAUSE_PROCESS** = `2`

The **Tween** will process regardless of whether SceneTree is paused.

---

enum **TransitionType**: 🔗

TransitionType **TRANS_LINEAR** = `0`

The animation is interpolated linearly.

TransitionType **TRANS_SINE** = `1`

The animation is interpolated using a sine function.

TransitionType **TRANS_QUINT** = `2`

The animation is interpolated with a quintic (to the power of 5) function.

TransitionType **TRANS_QUART** = `3`

The animation is interpolated with a quartic (to the power of 4) function.

TransitionType **TRANS_QUAD** = `4`

The animation is interpolated with a quadratic (to the power of 2) function.

TransitionType **TRANS_EXPO** = `5`

The animation is interpolated with an exponential (to the power of x) function.

TransitionType **TRANS_ELASTIC** = `6`

The animation is interpolated with elasticity, wiggling around the edges.

TransitionType **TRANS_CUBIC** = `7`

The animation is interpolated with a cubic (to the power of 3) function.

TransitionType **TRANS_CIRC** = `8`

The animation is interpolated with a function using square roots.

TransitionType **TRANS_BOUNCE** = `9`

The animation is interpolated by bouncing at the end.

TransitionType **TRANS_BACK** = `10`

The animation is interpolated backing out at ends.

TransitionType **TRANS_SPRING** = `11`

The animation is interpolated like a spring towards the end.

enum **EaseType**: 🔗

EaseType **EASE_IN** = `0`

The interpolation starts slowly and speeds up towards the end.

EaseType **EASE_OUT** = `1`

The interpolation starts quickly and slows down towards the end.

EaseType **EASE_IN_OUT** = `2`

A combination of EASE_IN and EASE_OUT. The interpolation is slowest at both ends.

EaseType **EASE_OUT_IN** = `3`

A combination of EASE_IN and EASE_OUT. The interpolation is fastest at both ends.

Method Descriptions

Tween **bind_node**(node: Node) 🔗

Binds this **Tween** with the given `node`. **Tween**s are processed directly by the SceneTree, so they run independently of the animated nodes. When you bind a Node with the **Tween**, the **Tween** will halt the animation when the object is not inside tree and the **Tween** will be automatically killed when the bound object is freed. Also TWEEN_PAUSE_BOUND will make the pausing behavior dependent on the bound node.

For a shorter way to create and bind a **Tween**, you can use Node.create_tween.

Tween **chain**() 🔗

Used to chain two Tweeners after set_parallel is called with `true`.

GDScript

```
var tween = create_tween().set_parallel(true)
tween.tween_property(...)
tween.tween_property(...) # Will run parallelly with above.
tween.chain().tween_property(...) # Will run after two above
are finished.
```

C#

```
Tween tween = CreateTween().SetParallel(true);
tween.TweenProperty(...);
tween.TweenProperty(...); // Will run parallelly with above.
tween.Chain().TweenProperty(...); // Will run after two above
are finished.
```

bool **custom_step**(delta: float)

Processes the **Tween** by the given `delta` value, in seconds. This is mostly useful for manual control when the **Tween** is paused. It can also be used to end the **Tween** animation immediately, by setting `delta` longer than the whole duration of the **Tween** animation.

Returns `true` if the **Tween** still has Tweeners that haven't finished.

int **get_loops_left**() const

Returns the number of remaining loops for this **Tween** (see set_loops). A return value of `-1` indicates an infinitely looping **Tween**, and a return value of `0` indicates that the **Tween** has already finished.

float **get_total_elapsed_time**() const

Returns the total time in seconds the **Tween** has been animating (i.e. the time since it started, not counting pauses etc.). The time is affected by set_speed_scale, and stop will reset it to `0`.

Note: As it results from accumulating frame deltas, the time returned after the **Tween** has finished animating will be slightly greater than the actual **Tween** duration.

Variant **interpolate_value**(initial_value: Variant, delta_value: Variant, elapsed_time: float, duration: float, trans_type: TransitionType, ease_type: EaseType) static 🔗

This method can be used for manual interpolation of a value, when you don't want **Tween** to do animating for you. It's similar to @GlobalScope.lerp, but with support for custom transition and easing.

`initial_value` is the starting value of the interpolation.

`delta_value` is the change of the value in the interpolation, i.e. it's equal to `final_value - initial_value`.

`elapsed_time` is the time in seconds that passed after the interpolation started and it's used to control the position of the interpolation. E.g. when it's equal to half of the `duration`, the interpolated value will be halfway between initial and final values. This value can also be greater than `duration` or lower than 0, which will extrapolate the value.

`duration` is the total time of the interpolation.

Note: If `duration` is equal to `0`, the method will always return the final value, regardless of `elapsed_time` provided.

---

bool **is_running**() 🔗

Returns whether the **Tween** is currently running, i.e. it wasn't paused and it's not finished.

---

bool **is_valid**() 🔗

Returns whether the **Tween** is valid. A valid **Tween** is a **Tween** contained by the scene tree (i.e. the array from SceneTree.get_processed_tweens will contain this **Tween**). A **Tween** might become invalid when it has finished tweening, is killed, or when created with `Tween.new()`. Invalid **Tweens** can't have Tweeners appended.

---

void **kill**() 🔗

Aborts all tweening operations and invalidates the **Tween**.

Tween **parallel**() 🔗

Makes the next Tweener run parallelly to the previous one.

Example:

GDScript

```
var tween = create_tween()
tween.tween_property(...)
tween.parallel().tween_property(...)
tween.parallel().tween_property(...)
```

C#

```
Tween tween = CreateTween();
tween.TweenProperty(...);
tween.Parallel().TweenProperty(...);
tween.Parallel().TweenProperty(...);
```

All Tweeners in the example will run at the same time.

You can make the **Tween** parallel by default by using set_parallel.

---

void **pause**()

Pauses the tweening. The animation can be resumed by using play.

Note: If a Tween is paused and not bound to any node, it will exist indefinitely until manually started or invalidated. If you lose a reference to such Tween, you can retrieve it using SceneTree.get_processed_tweens.

---

void **play**()

Resumes a paused or stopped **Tween**.

Tween **set_ease**(ease: EaseType)

Sets the default ease type for PropertyTweeners and MethodTweeners animated by this **Tween**.

If not specified, the default value is EASE_IN_OUT.

Tween **set_loops**(loops: int = 0)

Sets the number of times the tweening sequence will be repeated, i.e. `set_loops(2)` will run the animation twice.

Calling this method without arguments will make the **Tween** run infinitely, until either it is killed with kill, the **Tween**'s bound node is freed, or all the animated objects have been freed (which makes further animation impossible).

Warning: Make sure to always add some duration/delay when using infinite loops. To prevent the game freezing, 0-duration looped animations (e.g. a single CallbackTweener with no delay) are stopped after a small number of loops, which may produce unexpected results. If a **Tween**'s lifetime depends on some node, always use bind_node.

Tween **set_parallel**(parallel: bool = true) 🔗

If `parallel` is `true`, the Tweeners appended after this method will by default run simultaneously, as opposed to sequentially.

Note: Just like with parallel, the tweener added right before this method will also be part of the parallel step.

```
tween.tween_property(self, "position", Vector2(300, 0), 0.5)
tween.set_parallel()
tween.tween_property(self, "modulate", Color.GREEN, 0.5) # Runs together with the position tweener.
```

Tween **set_pause_mode**(mode: TweenPauseMode) 🔗

Determines the behavior of the **Tween** when the SceneTree is paused. Check TweenPauseMode for options.

Default value is TWEEN_PAUSE_BOUND.

Tween **set_process_mode**(mode: TweenProcessMode) 🔗

Determines whether the **Tween** should run after process frames (see Node._process) or physics frames (see Node._physics_process).

Default value is TWEEN_PROCESS_IDLE.

Tween **set_speed_scale**(speed: float) 🔗

Scales the speed of tweening. This affects all Tweeners and their delays.

Tween **set_trans**(trans: TransitionType)

Sets the default transition type for PropertyTweeners and MethodTweeners animated by this **Tween**.

If not specified, the default value is TRANS_LINEAR.

void **stop**()

Stops the tweening and resets the **Tween** to its initial state. This will not remove any appended Tweeners.

Note: If a Tween is stopped and not bound to any node, it will exist indefinitely until manually started or invalidated. If you lose a reference to such Tween, you can retrieve it using SceneTree.get_processed_tweens.

CallbackTweener **tween_callback**(callback: Callable)

Creates and appends a CallbackTweener. This method can be used to call an arbitrary method in any object. Use Callable.bind to bind additional arguments for the call.

Example: Object that keeps shooting every 1 second:

GDScript

```gdscript
var tween = get_tree().create_tween().set_loops()
tween.tween_callback(shoot).set_delay(1)
```

C#

```csharp
Tween tween = GetTree().CreateTween().SetLoops();
tween.TweenCallback(Callable.From(Shoot)).SetDelay(1.0f);
```

Example: Turning a sprite red and then blue, with 2 second delay:

GDScript

```
var tween = get_tree().create_tween()
tween.tween_callback($Sprite.set_modulate.bind(Color.RED)).set_delay(2)
tween.tween_callback($Sprite.set_modulate.bind(Color.BLUE)).set_delay(2)
```

C#

```
Tween tween = GetTree().CreateTween();
Sprite2D sprite = GetNode<Sprite2D>("Sprite");
tween.TweenCallback(Callable.From(() => sprite.Modulate = Colors.Red)).SetDelay(2.0f);
tween.TweenCallback(Callable.From(() => sprite.Modulate = Colors.Blue)).SetDelay(2.0f);
```

IntervalTweener **tween_interval**(time: float)

Creates and appends an IntervalTweener. This method can be used to create delays in the tween animation, as an alternative to using the delay in other Tweeners, or when there's no animation (in which case the **Tween** acts as a timer). `time` is the length of the interval, in seconds.

Example: Creating an interval in code execution:

GDScript

```
# ... some code
await create_tween().tween_interval(2).finished
# ... more code
```

C#

```
// ... some code
await ToSignal(CreateTween().TweenInterval(2.0f),
Tween.SignalName.Finished);
// ... more code
```

Example: Creating an object that moves back and forth and jumps every few seconds:

GDScript

```
var tween = create_tween().set_loops()
tween.tween_property($Sprite, "position:x", 200.0,
1).as_relative()
tween.tween_callback(jump)
tween.tween_interval(2)
tween.tween_property($Sprite, "position:x", -200.0,
1).as_relative()
tween.tween_callback(jump)
tween.tween_interval(2)
```

C#

```
Tween tween = CreateTween().SetLoops();
tween.TweenProperty(GetNode("Sprite"), "position:x", 200.0f,
1.0f).AsRelative();
tween.TweenCallback(Callable.From(Jump));
tween.TweenInterval(2.0f);
tween.TweenProperty(GetNode("Sprite"), "position:x", -200.0f,
1.0f).AsRelative();
tween.TweenCallback(Callable.From(Jump));
tween.TweenInterval(2.0f);
```

MethodTweener **tween_method**(method: Callable, from: Variant, to: Variant, duration: float) 🔗

Creates and appends a MethodTweener. This method is similar to a combination of tween_callback and tween_property. It calls a method over time with a tweened value provided as an argument. The value is tweened between `from` and `to` over the time specified by `duration`, in seconds. Use Callable.bind to bind additional arguments for the call. You can use MethodTweener.set_ease and MethodTweener.set_trans to tweak the easing and transition of the value or MethodTweener.set_delay to delay the tweening.

Example: Making a 3D object look from one point to another point:

GDScript

```
var tween = create_tween()
tween.tween_method(look_at.bind(Vector3.UP), Vector3(-1, 0,
-1), Vector3(1, 0, -1), 1) # The look_at() method takes up
vector as second argument.
```

C#

```
Tween tween = CreateTween();
tween.TweenMethod(Callable.From((Vector3 target) =>
LookAt(target, Vector3.Up)), new Vector3(-1.0f, 0.0f, -1.0f),
new Vector3(1.0f, 0.0f, -1.0f), 1.0f); // Use lambdas to bind
additional arguments for the call.
```

Example: Setting the text of a Label, using an intermediate method and after a delay:

GDScript

```
func _ready():
    var tween = create_tween()
    tween.tween_method(set_label_text, 0, 10, 1).set_delay(1)

func set_label_text(value: int):
    $Label.text = "Counting " + str(value)
```

C#

```
public override void _Ready()
{
    base._Ready();

    Tween tween = CreateTween();
    tween.TweenMethod(Callable.From<int>(SetLabelText), 0.0f,
10.0f, 1.0f).SetDelay(1.0f);
}

private void SetLabelText(int value)
{
    GetNode<Label>("Label").Text = $"Counting {value}";
}
```

PropertyTweener **tween_property**(object: Object, property: NodePath, final_val: Variant, duration: float) 🔗

Creates and appends a PropertyTweener. This method tweens a `property` of an `object` between an initial value and `final_val` in a span of time equal to `duration`, in seconds. The initial value by default is the property's value at the time the tweening of the PropertyTweener starts.

Example:

GDScript

```
var tween = create_tween()
tween.tween_property($Sprite, "position", Vector2(100, 200), 1)
tween.tween_property($Sprite, "position", Vector2(200, 300), 1)
```

C#

```
Tween tween = CreateTween();
tween.TweenProperty(GetNode("Sprite"), "position", new
Vector2(100.0f, 200.0f), 1.0f);
tween.TweenProperty(GetNode("Sprite"), "position", new
Vector2(200.0f, 300.0f), 1.0f);
```

will move the sprite to position (100, 200) and then to (200, 300). If you use PropertyTweener.from or PropertyTweener.from_current, the starting position will be overwritten by the given value instead. See other methods in PropertyTweener to see how the tweening can be tweaked further.

Note: You can find the correct property name by hovering over the property in the Inspector. You can also provide the components of a property directly by using `"property:component"` (eg. `position:x`), where it would only apply to that particular component.

Example: Moving an object twice from the same position, with different transition types:

GDScript

```
var tween = create_tween()
tween.tween_property($Sprite, "position", Vector2.RIGHT * 300,
1).as_relative().set_trans(Tween.TRANS_SINE)
tween.tween_property($Sprite, "position", Vector2.RIGHT * 300,
1).as_relative().from_current().set_trans(Tween.TRANS_EXPO)
```

C#

```
Tween tween = CreateTween();
tween.TweenProperty(GetNode("Sprite"), "position",
Vector2.Right * 300.0f,
1.0f).AsRelative().SetTrans(Tween.TransitionType.Sine);
tween.TweenProperty(GetNode("Sprite"), "position",
Vector2.Right * 300.0f,
1.0f).AsRelative().FromCurrent().SetTrans(Tween.TransitionType.
Expo);
```

Tweener

Inherits: RefCounted **<** Object

Inherited By: CallbackTweener, IntervalTweener, MethodTweener, PropertyTweener

Abstract class for all Tweeners used by Tween.

Description

Tweeners are objects that perform a specific animating task, e.g. interpolating a property or calling a method at a given time. A **Tweener** can't be created manually, you need to use a dedicated method from Tween.

Signals

finished() 🔗

Emitted when the **Tweener** has just finished its job.

UDPServer

Inherits: RefCounted **<** Object

Helper class to implement a UDP server.

Description

A simple server that opens a UDP socket and returns connected PacketPeerUDP upon receiving new packets. See also PacketPeerUDP.connect_to_host.

After starting the server (listen), you will need to poll it at regular intervals (e.g. inside Node._process) for it to process new packets, delivering them to the appropriate PacketPeerUDP, and taking new connections.

Below a small example of how it can be used:

GDScript

```
# server_node.gd
class_name ServerNode
extends Node

var server := UDPServer.new()
var peers = []

func _ready():
    server.listen(4242)

func _process(delta):
    server.poll() # Important!
    if server.is_connection_available():
        var peer: PacketPeerUDP = server.take_connection()
        var packet = peer.get_packet()
        print("Accepted peer: %s:%s" % [peer.get_packet_ip(),
peer.get_packet_port()])
        print("Received data: %s" %
[packet.get_string_from_utf8()])
        # Reply so it knows we received the message.
        peer.put_packet(packet)
        # Keep a reference so we can keep contacting the remote
peer.
        peers.append(peer)

    for i in range(0, peers.size()):
        pass # Do something with the connected peers.
```

C#

```
// ServerNode.cs
using Godot;
using System.Collections.Generic;

public partial class ServerNode : Node
{
    private UdpServer _server = new UdpServer();
    private List<PacketPeerUdp> _peers  = new
List<PacketPeerUdp>();

    public override void _Ready()
    {
        _server.Listen(4242);
    }

    public override void _Process(double delta)
    {
        _server.Poll(); // Important!
        if (_server.IsConnectionAvailable())
        {
            PacketPeerUdp peer = _server.TakeConnection();
            byte[] packet = peer.GetPacket();
            GD.Print($"Accepted Peer: {peer.GetPacketIP()}:
{peer.GetPacketPort()}");
            GD.Print($"Received Data:
{packet.GetStringFromUtf8()}");
            // Reply so it knows we received the message.
            peer.PutPacket(packet);
            // Keep a reference so we can keep contacting the
remote peer.
            _peers.Add(peer);
        }
        foreach (var peer in _peers)
        {
            // Do something with the peers.
        }
    }
}
```

GDScript

```
# client_node.gd
class_name ClientNode
extends Node

var udp := PacketPeerUDP.new()
var connected = false

func _ready():
    udp.connect_to_host("127.0.0.1", 4242)

func _process(delta):
    if !connected:
        # Try to contact server
        udp.put_packet("The answer is... 42!".to_utf8_buffer())
    if udp.get_available_packet_count() > 0:
        print("Connected: %s" %
udp.get_packet().get_string_from_utf8())
        connected = true
```

C#

```
// ClientNode.cs
using Godot;

public partial class ClientNode : Node
{
    private PacketPeerUdp _udp = new PacketPeerUdp();
    private bool _connected = false;

    public override void _Ready()
    {
        _udp.ConnectToHost("127.0.0.1", 4242);
    }

    public override void _Process(double delta)
    {
        if (!_connected)
        {
            // Try to contact server
            _udp.PutPacket("The Answer Is..42!".ToUtf8Buffer());
        }
        if (_udp.GetAvailablePacketCount() > 0)
        {
            GD.Print($"Connected: {_udp.GetPacket().GetStringFromUtf8()}");
            _connected = true;
        }
    }
}
```

Properties

int	max_pending_connections	`16`

Methods

int	get_local_port() const
bool	is_connection_available() const

bool	is_listening() const
Error	listen(port: int, bind_address: String = "*")
Error	poll()
void	stop()
PacketPeerUDP	take_connection()

Property Descriptions

int **max_pending_connections** = `16`

- void **set_max_pending_connections**(value: int)
- int **get_max_pending_connections**()

Define the maximum number of pending connections, during poll, any new pending connection exceeding that value will be automatically dropped. Setting this value to `0` effectively prevents any new pending connection to be accepted (e.g. when all your players have connected).

Method Descriptions

int **get_local_port**() const

Returns the local port this server is listening to.

---

bool **is_connection_available**() const 🔗

Returns `true` if a packet with a new address/port combination was received on the socket.

---

bool **is_listening**() const 🔗

Returns `true` if the socket is open and listening on a port.

Error **listen**(port: int, bind_address: String = "*") 🔗

Starts the server by opening a UDP socket listening on the given `port`. You can optionally specify a `bind_address` to only listen for packets sent to that address. See also PacketPeerUDP.bind.

Error **poll**() 🔗

Call this method at regular intervals (e.g. inside Node._process) to process new packets. And packet from known address/port pair will be delivered to the appropriate PacketPeerUDP, any packet received from an unknown address/port pair will be added as a pending connection (see is_connection_available, take_connection). The maximum number of pending connection is defined via max_pending_connections.

---

void **stop**() 🔗

Stops the server, closing the UDP socket if open. Will close all connected PacketPeerUDP accepted via take_connection (remote peers will not be notified).

PacketPeerUDP **take_connection**() 🔗

Returns the first pending connection (connected to the appropriate address/port). Will return `null` if no new connection is available. See also is_connection_available, PacketPeerUDP.connect_to_host.

UndoRedo

Inherits: Object

Provides a high-level interface for implementing undo and redo operations.

Description

UndoRedo works by registering methods and property changes inside "actions". You can create an action, then provide ways to do and undo this action using function calls and property changes, then commit the action.

When an action is committed, all of the `do_*` methods will run. If the undo method is used, the `undo_*` methods will run. If the redo method is used, once again, all of the `do_*` methods will run.

Here's an example on how to add an action:

GDScript

```
var undo_redo = UndoRedo.new()

func do_something():
    pass # Put your code here.

func undo_something():
    pass # Put here the code that reverts what's done by
"do_something()".

func _on_my_button_pressed():
    var node = get_node("MyNode2D")
    undo_redo.create_action("Move the node")
    undo_redo.add_do_method(do_something)
    undo_redo.add_undo_method(undo_something)
    undo_redo.add_do_property(node, "position",
Vector2(100,100))
    undo_redo.add_undo_property(node, "position",
node.position)
    undo_redo.commit_action()
```

C#

```
private UndoRedo _undoRedo;

public override void _Ready()
{
    _undoRedo = new UndoRedo();
}

public void DoSomething()
{
    // Put your code here.
}

public void UndoSomething()
{
    // Put here the code that reverts what's done by
"DoSomething()".
}

private void OnMyButtonPressed()
{
    var node = GetNode<Node2D>("MyNode2D");
    _undoRedo.CreateAction("Move the node");
    _undoRedo.AddDoMethod(new Callable(this,
MethodName.DoSomething));
    _undoRedo.AddUndoMethod(new Callable(this,
MethodName.UndoSomething));
    _undoRedo.AddDoProperty(node, "position", new Vector2(100,
100));
    _undoRedo.AddUndoProperty(node, "position", node.Position);
    _undoRedo.CommitAction();
}
```

Before calling any of the `add_(un)do_*` methods, you need to first call create_action. Afterwards you need to call commit_action.

If you don't need to register a method, you can leave add_do_method and add_undo_method out; the same goes for properties. You can also register more than one method/property.

If you are making an EditorPlugin and want to integrate into the editor's undo history, use EditorUndoRedoManager instead.

If you are registering multiple properties/method which depend on one another, be aware that by default undo operation are called in the same order they have been added. Therefore instead of grouping do operation with their undo operations it is better to group do on one side and undo on the other as shown below.

GDScript

```
undo_redo.create_action("Add object")

# DO
undo_redo.add_do_method(_create_object)
undo_redo.add_do_method(_add_object_to_singleton)

# UNDO
undo_redo.add_undo_method(_remove_object_from_singleton)
undo_redo.add_undo_method(_destroy_that_object)

undo_redo.commit_action()
```

C#

```
_undo_redo.CreateAction("Add object");

// DO
_undo_redo.AddDoMethod(new Callable(this,
MethodName.CreateObject));
_undo_redo.AddDoMethod(new Callable(this,
MethodName.AddObjectToSingleton));
```

```
// UNDO
_undo_redo.AddUndoMethod(new Callable(this,
MethodName.RemoveObjectFromSingleton));
_undo_redo.AddUndoMethod(new Callable(this,
MethodName.DestroyThatObject));

_undo_redo.CommitAction();
```

Properties

int	max_steps	`0`

Methods

void	add_do_method(callable: Callable)
void	add_do_property(object: Object, property: StringName, value: Variant)
void	add_do_reference(object: Object)
void	add_undo_method(callable: Callable)
void	add_undo_property(object: Object, property: StringName, value: Variant)

void	add_undo_reference(object: Object)
void	clear_history(increase_version: bool = true)
void	commit_action(execute: bool = true)
void	create_action(name: String, merge_mode: MergeMode = 0, backward_undo_ops: bool = false)
void	end_force_keep_in_merge_ends()
String	get_action_name(id: int)
int	get_current_action()
String	get_current_action_name() const
int	get_history_count()
int	get_version() const
bool	has_redo() const
bool	has_undo() const

bool	is_committing_action() const
bool	redo()
void	start_force_keep_in_merge_ends()
bool	undo()

Signals

version_changed()

Called when undo or redo was called.

Enumerations

enum **MergeMode**:

MergeMode **MERGE_DISABLE** = `0`

Makes "do"/"undo" operations stay in separate actions.

MergeMode **MERGE_ENDS** = `1`

Merges this action with the previous one if they have the same name. Keeps only the first action's "undo" operations and the last action's "do" operations. Useful for sequential changes to a single value.

MergeMode **MERGE_ALL** = `2`

Merges this action with the previous one if they have the same name.

Property Descriptions

int **max_steps** = `0` 🔗

- void **set_max_steps**(value: int)
- int **get_max_steps**()

The maximum number of steps that can be stored in the undo/redo history. If the number of stored steps exceeds this limit, older steps are removed from history and can no longer be reached by calling undo. A value of `0` or lower means no limit.

Method Descriptions

void **add_do_method**(callable: Callable) 🔗

Register a Callable that will be called when the action is committed.

---

void **add_do_property**(object: Object, property: StringName, value: Variant) 🔗

Register a `property` that would change its value to `value` when the action is committed.

---

void **add_do_reference**(object: Object) 🔗

Register a reference to an object that will be erased if the "do" history is deleted. This is useful for objects added by the "do" action and removed by the "undo" action.

When the "do" history is deleted, if the object is a RefCounted, it will be unreferenced. Otherwise, it will be freed. Do not use for resources.

```
var node = Node2D.new()
undo_redo.create_action("Add node")
undo_redo.add_do_method(add_child.bind(node))
undo_redo.add_do_reference(node)
undo_redo.add_undo_method(remove_child.bind(node))
undo_redo.commit_action()
```

---

void **add_undo_method**(callable: Callable)

Register a Callable that will be called when the action is undone.

---

void **add_undo_property**(object: Object, property: StringName, value: Variant)

Register a `property` that would change its value to `value` when the action is undone.

---

void **add_undo_reference**(object: Object)

Register a reference to an object that will be erased if the "undo" history is deleted. This is useful for objects added by the "undo" action and removed by the "do" action.

When the "undo" history is deleted, if the object is a RefCounted, it will be unreferenced. Otherwise, it will be freed. Do not use for resources.

```
var node = $Node2D
undo_redo.create_action("Remove node")
undo_redo.add_do_method(remove_child.bind(node))
undo_redo.add_undo_method(add_child.bind(node))
undo_redo.add_undo_reference(node)
undo_redo.commit_action()
```

---

void **clear_history**(increase_version: bool = true)

Clear the undo/redo history and associated references.

Passing `false` to `increase_version` will prevent the version number from increasing when the history is cleared.

---

void **commit_action**(execute: bool = true)

Commit the action. If `execute` is `true` (which it is by default), all "do" methods/properties are called/set when this function is called.

---

void **create_action**(name: String, merge_mode: MergeMode = 0, backward_undo_ops: bool = false)

Create a new action. After this is called, do all your calls to add_do_method, add_undo_method, add_do_property, and add_undo_property, then commit the action with commit_action.

The way actions are merged is dictated by `merge_mode`. See MergeMode for details.

The way undo operation are ordered in actions is dictated by `backward_undo_ops`. When `backward_undo_ops` is `false` undo option are ordered in the same order they were added. Which means the first operation to be added will be the first to be undone.

---

void **end_force_keep_in_merge_ends**() 🔗

Stops marking operations as to be processed even if the action gets merged with another in the MERGE_ENDS mode. See start_force_keep_in_merge_ends.

String **get_action_name**(id: int) 🔗

Gets the action name from its index.

---

int **get_current_action**() 🔗

Gets the index of the current action.

String **get_current_action_name**() const 🔗

Gets the name of the current action, equivalent to `get_action_name(get_current_action())`.

---

int **get_history_count**() 🔗

Returns how many elements are in the history.

---

int **get_version**() const 🔗

Gets the version. Every time a new action is committed, the **UndoRedo**'s version number is increased automatically.

This is useful mostly to check if something changed from a saved version.

---

bool **has_redo**() const

Returns `true` if a "redo" action is available.

bool **has_undo**() const

Returns `true` if an "undo" action is available.

bool **is_committing_action**() const

Returns `true` if the **UndoRedo** is currently committing the action, i.e. running its "do" method or property change (see commit_action).

bool **redo**()

Redo the last action.

void **start_force_keep_in_merge_ends**()

Marks the next "do" and "undo" operations to be processed even if the action gets merged with another in the MERGE_ENDS mode. Return to normal operation using end_force_keep_in_merge_ends.

bool **undo**()

Undo the last action.

UniformSetCacheRD

Inherits: Object

Uniform set cache manager for Rendering Device based renderers.

Description

Uniform set cache manager for Rendering Device based renderers. Provides a way to create a uniform set and reuse it in subsequent calls for as long as the uniform set exists. Uniform set will automatically be cleaned up when dependent objects are freed.

Methods

RID	get_cache(shader: RID, set: int, uniforms: Array[RDUniform]) static

---

Method Descriptions

RID **get_cache**(shader: RID, set: int, uniforms: Array[RDUniform]) static 🔗

Creates/returns a cached uniform set based on the provided uniforms for a given shader.

UPNP

Inherits: RefCounted **<** Object

Universal Plug and Play (UPnP) functions for network device discovery, querying and port forwarding.

Description

This class can be used to discover compatible UPNPDevices on the local network and execute commands on them, like managing port mappings (for port forwarding/NAT traversal) and querying the local and remote network IP address. Note that methods on this class are synchronous and block the calling thread.

To forward a specific port (here `7777`, note both discover and add_port_mapping can return errors that should be checked):

```
var upnp = UPNP.new()
upnp.discover()
upnp.add_port_mapping(7777)
```

To close a specific port (e.g. after you have finished using it):

```
upnp.delete_port_mapping(port)
```

Note: UPnP discovery blocks the current thread. To perform discovery without blocking the main thread, use Threads like this:

```
# Emitted when UPnP port mapping setup is completed (regardless
of success or failure).
signal upnp_completed(error)

# Replace this with your own server port number between 1024
and 65535.
```

```
const SERVER_PORT = 3928
var thread = null

func _upnp_setup(server_port):
    # UPNP queries take some time.
    var upnp = UPNP.new()
    var err = upnp.discover()

    if err != OK:
        push_error(str(err))
        emit_signal("upnp_completed", err)
        return

    if upnp.get_gateway() and upnp.get_gateway().is_valid_gateway():
        upnp.add_port_mapping(server_port, server_port, ProjectSettings.get_setting("application/config/name"), "UDP")
        upnp.add_port_mapping(server_port, server_port, ProjectSettings.get_setting("application/config/name"), "TCP")
        emit_signal("upnp_completed", OK)

func _ready():
    thread = Thread.new()
    thread.start(_upnp_setup.bind(SERVER_PORT))

func _exit_tree():
    # Wait for thread finish here to handle game exit while the thread is running.
    thread.wait_to_finish()
```

Terminology: In the context of UPnP networking, "gateway" (or "internet gateway device", short IGD) refers to network devices that allow computers in the local network to access the internet ("wide area network", WAN). These gateways are often also called "routers".

Pitfalls:

- As explained above, these calls are blocking and shouldn't be run on the main thread, especially as they can block for multiple seconds at a time. Use threading!

- Networking is physical and messy. Packets get lost in transit or get filtered, addresses, free ports and assigned mappings change, and devices may leave or join the network at any time. Be mindful of this, be diligent when checking and handling errors, and handle these gracefully if you can: add clear error UI, timeouts and re-try handling.
- Port mappings may change (and be removed) at any time, and the remote/external IP address of the gateway can change likewise. You should consider re-querying the external IP and try to update/refresh the port mapping periodically (for example, every 5 minutes and on networking failures).
- Not all devices support UPnP, and some users disable UPnP support. You need to handle this (e.g. documenting and requiring the user to manually forward ports, or adding alternative methods of NAT traversal, like a relay/mirror server, or NAT hole punching, STUN/TURN, etc.).
- Consider what happens on mapping conflicts. Maybe multiple users on the same network would like to play your game at the same time, or maybe another application uses the same port. Make the port configurable, and optimally choose a port automatically (re-trying with a different port on failure).

Further reading: If you want to know more about UPnP (and the Internet Gateway Device (IGD) and Port Control Protocol (PCP) specifically), Wikipedia [https://en.wikipedia.org/wiki/Universal_Plug_and_Play] is a good first stop, the specification can be found at the Open Connectivity Foundation [https://openconnectivity.org/developer/specifications/upnp-resources/upnp/] and Godot's implementation is based on the MiniUPnP client [https://github.com/miniupnp/miniupnp].

Properties

bool	discover_ipv6	`false`
int	discover_local_port	`0`
String	discover_multicast_if	`""`

Methods

void	add_device(device: UPNPDevice)
int	add_port_mapping(port: int, port_internal: int = 0, desc: String = "", proto: String = "UDP", duration: int = 0) const
void	clear_devices()
int	delete_port_mapping(port: int, proto: String = "UDP") const
int	discover(timeout: int = 2000, ttl: int = 2, device_filter: String = "InternetGatewayDevice")
UPNPDevice	get_device(index: int) const
int	get_device_count() const

UPNPDevice	get_gateway() const
String	query_external_address() const
void	remove_device(index: int)
void	set_device(index: int, device: UPNPDevice)

Enumerations

enum **UPNPResult**: 🔗

UPNPResult **UPNP_RESULT_SUCCESS** = `0`

UPNP command or discovery was successful.

UPNPResult **UPNP_RESULT_NOT_AUTHORIZED** = `1`

Not authorized to use the command on the UPNPDevice. May be returned when the user disabled UPNP on their router.

UPNPResult **UPNP_RESULT_PORT_MAPPING_NOT_FOUND** = `2`

No port mapping was found for the given port, protocol combination on the given UPNPDevice.

UPNPResult **UPNP_RESULT_INCONSISTENT_PARAMETERS** = `3`

Inconsistent parameters.

UPNPResult **UPNP_RESULT_NO_SUCH_ENTRY_IN_ARRAY** = `4`

No such entry in array. May be returned if a given port, protocol combination is not found on an UPNPDevice.

UPNPResult **UPNP_RESULT_ACTION_FAILED** = `5`

The action failed.

UPNPResult **UPNP_RESULT_SRC_IP_WILDCARD_NOT_PERMITTED** = `6`

The UPNPDevice does not allow wildcard values for the source IP address.

UPNPResult **UPNP_RESULT_EXT_PORT_WILDCARD_NOT_PERMITTED** = `7`

The UPNPDevice does not allow wildcard values for the external port.

UPNPResult **UPNP_RESULT_INT_PORT_WILDCARD_NOT_PERMITTED** = `8`

The UPNPDevice does not allow wildcard values for the internal port.

UPNPResult **UPNP_RESULT_REMOTE_HOST_MUST_BE_WILDCARD** = `9`

The remote host value must be a wildcard.

UPNPResult **UPNP_RESULT_EXT_PORT_MUST_BE_WILDCARD** = `10`

The external port value must be a wildcard.

UPNPResult **UPNP_RESULT_NO_PORT_MAPS_AVAILABLE** = `11`

No port maps are available. May also be returned if port mapping functionality is not available.

UPNPResult **UPNP_RESULT_CONFLICT_WITH_OTHER_MECHANISM** = `12`

Conflict with other mechanism. May be returned instead of UPNP_RESULT_CONFLICT_WITH_OTHER_MAPPING if a port mapping conflicts with an existing one.

UPNPResult **UPNP_RESULT_CONFLICT_WITH_OTHER_MAPPING** = `13`

Conflict with an existing port mapping.

UPNPResult **UPNP_RESULT_SAME_PORT_VALUES_REQUIRED** = `14`

External and internal port values must be the same.

UPNPResult **UPNP_RESULT_ONLY_PERMANENT_LEASE_SUPPORTED** = `15`

Only permanent leases are supported. Do not use the `duration` parameter when adding port mappings.

UPNPResult **UPNP_RESULT_INVALID_GATEWAY** = `16`

Invalid gateway.

UPNPResult **UPNP_RESULT_INVALID_PORT** = `17`

Invalid port.

UPNPResult **UPNP_RESULT_INVALID_PROTOCOL** = `18`

Invalid protocol.

UPNPResult **UPNP_RESULT_INVALID_DURATION** = `19`

Invalid duration.

UPNPResult **UPNP_RESULT_INVALID_ARGS** = `20`

Invalid arguments.

UPNPResult **UPNP_RESULT_INVALID_RESPONSE** = `21`

Invalid response.

UPNPResult **UPNP_RESULT_INVALID_PARAM** = `22`

Invalid parameter.

UPNPResult **UPNP_RESULT_HTTP_ERROR** = `23`

HTTP error.

UPNPResult **UPNP_RESULT_SOCKET_ERROR** = `24`

Socket error.

UPNPResult **UPNP_RESULT_MEM_ALLOC_ERROR** = `25`

Error allocating memory.

UPNPResult **UPNP_RESULT_NO_GATEWAY** = `26`

No gateway available. You may need to call discover first, or discovery didn't detect any valid IGDs (InternetGatewayDevices).

UPNPResult **UPNP_RESULT_NO_DEVICES** = `27`

No devices available. You may need to call discover first, or discovery didn't detect any valid UPNPDevices.

UPNPResult **UPNP_RESULT_UNKNOWN_ERROR** = `28`

Unknown error.

Property Descriptions

bool **discover_ipv6** = `false`

- void **set_discover_ipv6**(value: bool)
- bool **is_discover_ipv6**()

If `true`, IPv6 is used for UPNPDevice discovery.

---

int **discover_local_port** = `0`

- void **set_discover_local_port**(value: int)
- int **get_discover_local_port**()

If `0`, the local port to use for discovery is chosen automatically by the system. If `1`, discovery will be done from the source port 1900 (same as destination port). Otherwise, the value will be used as the port.

String **discover_multicast_if** = `""`

- void **set_discover_multicast_if**(value: String)
- String **get_discover_multicast_if**()

Multicast interface to use for discovery. Uses the default multicast interface if empty.

Method Descriptions

void **add_device**(device: UPNPDevice) 🔗

Adds the given UPNPDevice to the list of discovered devices.

---

int **add_port_mapping**(port: int, port_internal: int = 0, desc: String = "", proto: String = "UDP", duration: int = 0) const 🔗

Adds a mapping to forward the external `port` (between 1 and 65535, although recommended to use port 1024 or above) on the default gateway (see get_gateway) to the `port_internal` on the local machine for the given protocol `proto` (either `"TCP"` or `"UDP"`, with UDP being the default). If a port mapping for the given port and protocol combination already exists on that gateway device, this method tries to overwrite it. If that is not desired, you can retrieve the gateway manually with get_gateway and call add_port_mapping on it, if any. Note that forwarding a well-known port (below 1024) with UPnP may fail depending on the device.

Depending on the gateway device, if a mapping for that port already exists, it will either be updated or it will refuse this command due to that conflict, especially if the existing mapping for that port wasn't created via UPnP or points to a different network address (or device) than this one.

If `port_internal` is `0` (the default), the same port number is used for both the external and the internal port (the `port` value).

The description (`desc`) is shown in some routers management UIs and can be used to point out which application added the mapping.

The mapping's lease `duration` can be limited by specifying a duration in seconds. The default of `0` means no duration, i.e. a permanent lease and notably some devices only support these permanent leases. Note that whether permanent or not, this is only a request and the gateway may still decide at any point to remove the mapping (which usually happens on a reboot of the gateway, when its external IP address changes, or on some models when it detects a port mapping has become inactive, i.e. had no traffic for multiple minutes). If not `0` (permanent), the allowed range according to spec is between `120` (2 minutes) and `86400` seconds (24 hours).

See UPNPResult for possible return values.

void **clear_devices**()

Clears the list of discovered devices.

int **delete_port_mapping**(port: int, proto: String = "UDP") const

Deletes the port mapping for the given port and protocol combination on the default gateway (see get_gateway) if one exists. `port` must be a valid port between 1 and 65535, `proto` can be either `"TCP"` or `"UDP"`. May be refused for mappings pointing to addresses other than this one, for well-known ports (below 1024), or for mappings not added via UPnP. See UPNPResult for possible return values.

int **discover**(timeout: int = 2000, ttl: int = 2, device_filter: String = "InternetGatewayDevice")

Discovers local UPNPDevices. Clears the list of previously discovered devices.

Filters for IGD (InternetGatewayDevice) type devices by default, as those manage port forwarding. `timeout` is the time to wait for responses in milliseconds. `ttl` is the time-to-live; only touch this if you know what you're doing.

See UPNPResult for possible return values.

UPNPDevice **get_device**(index: int) const

Returns the UPNPDevice at the given `index`.

---

int **get_device_count**() const

Returns the number of discovered UPNPDevices.

UPNPDevice **get_gateway**() const

Returns the default gateway. That is the first discovered UPNPDevice that is also a valid IGD (InternetGatewayDevice).

String **query_external_address**() const

Returns the external IP address of the default gateway (see get_gateway) as string. Returns an empty string on error.

---

void **remove_device**(index: int)

Removes the device at `index` from the list of discovered devices.

---

void **set_device**(index: int, device: UPNPDevice)

Sets the device at `index` from the list of discovered devices to `device`.

UPNPDevice

Inherits: RefCounted **<** Object

Universal Plug and Play (UPnP) device.

Description

Universal Plug and Play (UPnP) device. See UPNP for UPnP discovery and utility functions. Provides low-level access to UPNP control commands. Allows to manage port mappings (port forwarding) and to query network information of the device (like local and external IP address and status). Note that methods on this class are synchronous and block the calling thread.

Properties

String	description_url	""
String	igd_control_url	""
String	igd_our_addr	""
String	igd_service_type	""
IGDStatus	igd_status	9

String	service_type	""

Methods

int	add_port_mapping(port: int, port_internal: int = 0, desc: String = "", proto: String = "UDP", duration: int = 0) const
int	delete_port_mapping(port: int, proto: String = "UDP") const
bool	is_valid_gateway() const
String	query_external_address() const

Enumerations

enum **IGDStatus**:

IGDStatus **IGD_STATUS_OK** = `0`

OK.

IGDStatus **IGD_STATUS_HTTP_ERROR** = `1`

HTTP error.

IGDStatus **IGD_STATUS_HTTP_EMPTY** = `2`

Empty HTTP response.

IGDStatus **IGD_STATUS_NO_URLS** = `3`

Deprecated: This value is no longer used.

Returned response contained no URLs.

IGDStatus **IGD_STATUS_NO_IGD** = `4`

Not a valid IGD.

IGDStatus **IGD_STATUS_DISCONNECTED** = `5`

Disconnected.

IGDStatus **IGD_STATUS_UNKNOWN_DEVICE** = `6`

Unknown device.

IGDStatus **IGD_STATUS_INVALID_CONTROL** = `7`

Invalid control.

IGDStatus **IGD_STATUS_MALLOC_ERROR** = `8`

Deprecated: This value is no longer used.

Memory allocation error.

IGDStatus **IGD_STATUS_UNKNOWN_ERROR** = `9`

Unknown error.

Property Descriptions

String **description_url** = `""` 🔗

- void **set_description_url**(value: String)
- String **get_description_url**()

URL to the device description.

String **igd_control_url** = `""`

- void **set_igd_control_url**(value: String)
- String **get_igd_control_url**()

IDG control URL.

String **igd_our_addr** = `""`

- void **set_igd_our_addr**(value: String)
- String **get_igd_our_addr**()

Address of the local machine in the network connecting it to this **UPNPDevice**.

String **igd_service_type** = `""`

- void **set_igd_service_type**(value: String)
- String **get_igd_service_type**()

IGD service type.

IGDStatus **igd_status** = `9`

- void **set_igd_status**(value: IGDStatus)
- IGDStatus **get_igd_status**()

IGD status. See IGDStatus.

String **service_type** = `""`

- void **set_service_type**(value: String)
- String **get_service_type**()

Service type.

Method Descriptions

int **add_port_mapping**(port: int, port_internal: int = 0, desc: String = "", proto: String = "UDP", duration: int = 0) const

Adds a port mapping to forward the given external port on this **UPNPDevice** for the given protocol to the local machine. See UPNP.add_port_mapping.

int **delete_port_mapping**(port: int, proto: String = "UDP") const

Deletes the port mapping identified by the given port and protocol combination on this device. See UPNP.delete_port_mapping.

bool **is_valid_gateway**() const

Returns `true` if this is a valid IGD (InternetGatewayDevice) which potentially supports port forwarding.

String **query_external_address**() const

Returns the external IP address of this **UPNPDevice** or an empty string.

WeakRef

Inherits: RefCounted **<** Object

Holds an Object. If the object is RefCounted, it doesn't update the reference count.

Description

A weakref can hold a RefCounted without contributing to the reference counter. A weakref can be created from an Object using @GlobalScope.weakref. If this object is not a reference, weakref still works, however, it does not have any effect on the object. Weakrefs are useful in cases where multiple classes have variables that refer to each other. Without weakrefs, using these classes could lead to memory leaks, since both references keep each other from being released. Making part of the variables a weakref can prevent this cyclic dependency, and allows the references to be released.

Methods

Variant	get_ref() const

Method Descriptions

Variant **get_ref**() const 🔗

Returns the Object this weakref is referring to. Returns `null` if that object no longer exists.

WebRTCDataChannel

Inherits: PacketPeer **<** RefCounted **<** Object

Inherited By: WebRTCDataChannelExtension

There is currently no description for this class. Please help us by contributing one!

Properties

WriteMode	write_mode	`1`

Methods

void	close()
int	get_buffered_amount() const
int	get_id() const
String	get_label() const
int	get_max_packet_life_time() const

int	get_max_retransmits() const
String	get_protocol() const
ChannelState	get_ready_state() const
bool	is_negotiated() const
bool	is_ordered() const
Error	poll()
bool	was_string_packet() const

Enumerations

enum **WriteMode**: 🔗

WriteMode **WRITE_MODE_TEXT** = `0`

Tells the channel to send data over this channel as text. An external peer (non-Godot) would receive this as a string.

WriteMode **WRITE_MODE_BINARY** = `1`

Tells the channel to send data over this channel as binary. An external peer (non-Godot) would receive this as array buffer or blob.

---

enum **ChannelState**:

ChannelState **STATE_CONNECTING** = `0`

The channel was created, but it's still trying to connect.

ChannelState **STATE_OPEN** = `1`

The channel is currently open, and data can flow over it.

ChannelState **STATE_CLOSING** = `2`

The channel is being closed, no new messages will be accepted, but those already in queue will be flushed.

ChannelState **STATE_CLOSED** = `3`

The channel was closed, or connection failed.

Property Descriptions

WriteMode **write_mode** = `1`

- void **set_write_mode**(value: WriteMode)
- WriteMode **get_write_mode**()

The transfer mode to use when sending outgoing packet. Either text or binary.

Method Descriptions

void **close**()

Closes this data channel, notifying the other peer.

---

int **get_buffered_amount**() const 🔗

Returns the number of bytes currently queued to be sent over this channel.

---

int **get_id**() const 🔗

Returns the ID assigned to this channel during creation (or auto-assigned during negotiation).

If the channel is not negotiated out-of-band the ID will only be available after the connection is established (will return `65535` until then).

String **get_label**() const 🔗

Returns the label assigned to this channel during creation.

---

int **get_max_packet_life_time**() const 🔗

Returns the `maxPacketLifeTime` value assigned to this channel during creation.

Will be `65535` if not specified.

---

int **get_max_retransmits**() const 🔗

Returns the `maxRetransmits` value assigned to this channel during creation.

Will be `65535` if not specified.

String **get_protocol**() const

Returns the sub-protocol assigned to this channel during creation. An empty string if not specified.

ChannelState **get_ready_state**() const

Returns the current state of this channel, see ChannelState.

---

bool **is_negotiated**() const

Returns `true` if this channel was created with out-of-band configuration.

---

bool **is_ordered**() const

Returns `true` if this channel was created with ordering enabled (default).

Error **poll**()

Reserved, but not used for now.

---

bool **was_string_packet**() const

Returns `true` if the last received packet was transferred as text. See write_mode.

WebRTCDataChannelExtension

Inherits: WebRTCDataChannel **<** PacketPeer **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

Methods

void	_close() virtual
int	_get_available_packet_count() virtual const
int	_get_buffered_amount() virtual const
int	_get_id() virtual const
String	_get_label() virtual const
int	_get_max_packet_life_time() virtual const
int	_get_max_packet_size() virtual const

int	_get_max_retransmits() virtual const
Error	_get_packet(r_buffer: `const uint8_t **`, r_buffer_size: `int32_t*`) virtual
String	_get_protocol() virtual const
ChannelState	_get_ready_state() virtual const
WriteMode	_get_write_mode() virtual const
bool	_is_negotiated() virtual const
bool	_is_ordered() virtual const
Error	_poll() virtual
Error	_put_packet(p_buffer: `const uint8_t*`, p_buffer_size: int) virtual
void	_set_write_mode(p_write_mode: WriteMode) virtual
bool	_was_string_packet() virtual const

Method Descriptions

void **_close**() virtual 🔗

There is currently no description for this method. Please help us by contributing one!

---

int **_get_available_packet_count**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

int **_get_buffered_amount**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

int **_get_id**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

String **_get_label**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

int **_get_max_packet_life_time**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

int **_get_max_packet_size**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

int **_get_max_retransmits**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Error **_get_packet**(r_buffer: `const uint8_t **`, r_buffer_size: `int32_t*`) virtual 🔗

There is currently no description for this method. Please help us by contributing one!

String **_get_protocol**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

ChannelState **_get_ready_state**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

WriteMode **_get_write_mode**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_is_negotiated**() virtual const

There is currently no description for this method. Please help us by contributing one!

---

bool **_is_ordered**() virtual const

There is currently no description for this method. Please help us by contributing one!

Error **_poll**() virtual

There is currently no description for this method. Please help us by contributing one!

Error **_put_packet**(p_buffer: `const uint8_t*`, p_buffer_size: int) virtual

There is currently no description for this method. Please help us by contributing one!

---

void **_set_write_mode**(p_write_mode: WriteMode) virtual

There is currently no description for this method. Please help us by contributing one!

---

bool **_was_string_packet**() virtual const

There is currently no description for this method. Please help us by contributing one!

WebRTCMultiplayerPeer

Inherits: MultiplayerPeer **<** PacketPeer **<** RefCounted **<** Object

A simple interface to create a peer-to-peer mesh network composed of WebRTCPeerConnection that is compatible with the MultiplayerAPI.

Description

This class constructs a full mesh of WebRTCPeerConnection (one connection for each peer) that can be used as a MultiplayerAPI.multiplayer_peer.

You can add each WebRTCPeerConnection via add_peer or remove them via remove_peer. Peers must be added in WebRTCPeerConnection.STATE_NEW state to allow it to create the appropriate channels. This class will not create offers nor set descriptions, it will only poll them, and notify connections and disconnections.

When creating the peer via create_client or create_server the MultiplayerPeer.is_server_relay_supported method will return `true` enabling peer exchange and packet relaying when supported by the MultiplayerAPI implementation.

Note: When exporting to Android, make sure to enable the `INTERNET` permission in the Android export preset before exporting the project or using one-click deploy. Otherwise, network communication of any kind will be blocked by Android.

Methods

Error	add_peer(peer: WebRTCPeerConnection, peer_id: int, unreliable_lifetime: int = 1)
Error	create_client(peer_id: int, channels_config: Array = [])
Error	create_mesh(peer_id: int, channels_config: Array = [])
Error	create_server(channels_config: Array = [])
Dictionary	get_peer(peer_id: int)
Dictionary	get_peers()
bool	has_peer(peer_id: int)
void	remove_peer(peer_id: int)

Method Descriptions

Error **add_peer**(peer: WebRTCPeerConnection, peer_id: int, unreliable_lifetime: int = 1) 🔗

Add a new peer to the mesh with the given `peer_id`. The WebRTCPeerConnection must be in state WebRTCPeerConnection.STATE_NEW.

Three channels will be created for reliable, unreliable, and ordered transport. The value of `unreliable_lifetime` will be passed to the `"maxPacketLifetime"` option when creating unreliable and ordered channels (see WebRTCPeerConnection.create_data_channel).

Error **create_client**(peer_id: int, channels_config: Array = []) 🔗

Initialize the multiplayer peer as a client with the given `peer_id` (must be between 2 and 2147483647). In this mode, you should only call add_peer once and with `peer_id` of `1`. This mode enables MultiplayerPeer.is_server_relay_supported, allowing the upper MultiplayerAPI layer to perform peer exchange and packet relaying.

You can optionally specify a `channels_config` array of TransferMode which will be used to create extra channels (WebRTC only supports one transfer mode per channel).

Error **create_mesh**(peer_id: int, channels_config: Array = []) 🔗

Initialize the multiplayer peer as a mesh (i.e. all peers connect to each other) with the given `peer_id` (must be between 1 and 2147483647).

Error **create_server**(channels_config: Array = []) 🔗

Initialize the multiplayer peer as a server (with unique ID of `1`). This mode enables MultiplayerPeer.is_server_relay_supported, allowing the upper MultiplayerAPI layer to perform peer exchange and packet relaying.

You can optionally specify a `channels_config` array of TransferMode which will be used to create extra channels (WebRTC only supports one transfer mode per channel).

Dictionary **get_peer**(peer_id: int)

Returns a dictionary representation of the peer with given `peer_id` with three keys. `"connection"` containing the WebRTCPeerConnection to this peer, `"channels"` an array of three WebRTCDataChannel, and `"connected"` a boolean representing if the peer connection is currently connected (all three channels are open).

Dictionary **get_peers**()

Returns a dictionary which keys are the peer ids and values the peer representation as in get_peer.

---

bool **has_peer**(peer_id: int)

Returns `true` if the given `peer_id` is in the peers map (it might not be connected though).

---

void **remove_peer**(peer_id: int)

Remove the peer with given `peer_id` from the mesh. If the peer was connected, and MultiplayerPeer.peer_connected was emitted for it, then MultiplayerPeer.peer_disconnected will be emitted.

WebRTCPeerConnection

Inherits: RefCounted **<** Object

Inherited By: WebRTCPeerConnectionExtension

Interface to a WebRTC peer connection.

Description

A WebRTC connection between the local computer and a remote peer. Provides an interface to connect, maintain and monitor the connection.

Setting up a WebRTC connection between two peers may not seem a trivial task, but it can be broken down into 3 main steps:

- The peer that wants to initiate the connection (`A` from now on) creates an offer and send it to the other peer (`B` from now on).
- `B` receives the offer, generate and answer, and sends it to `A`).
- `A` and `B` then generates and exchange ICE candidates with each other.

After these steps, the connection should become connected. Keep on reading or look into the tutorial for more information.

Methods

Error	add_ice_candidate(media: String, index: int, name: String)

void	close()
WebRTCDataChannel	create_data_channel(label: String, options: Dictionary = {})
Error	create_offer()
ConnectionState	get_connection_state() const
GatheringState	get_gathering_state() const
SignalingState	get_signaling_state() const
Error	initialize(configuration: Dictionary = {})
Error	poll()
void	set_default_extension(extension_class: StringName) static
Error	set_local_description(type: String, sdp: String)
Error	set_remote_description(type: String, sdp: String)

Signals

data_channel_received(channel: WebRTCDataChannel)

Emitted when a new in-band channel is received, i.e. when the channel was created with `negotiated: false` (default).

The object will be an instance of WebRTCDataChannel. You must keep a reference of it or it will be closed automatically. See create_data_channel.

---

ice_candidate_created(media: String, index: int, name: String)

Emitted when a new ICE candidate has been created. The three parameters are meant to be passed to the remote peer over the signaling server.

---

session_description_created(type: String, sdp: String)

Emitted after a successful call to create_offer or set_remote_description (when it generates an answer). The parameters are meant to be passed to set_local_description on this object, and sent to the remote peer over the signaling server.

Enumerations

enum **ConnectionState**:

ConnectionState **STATE_NEW** = `0`

The connection is new, data channels and an offer can be created in this state.

ConnectionState **STATE_CONNECTING** = `1`

The peer is connecting, ICE is in progress, none of the transports has failed.

ConnectionState **STATE_CONNECTED** = `2`

The peer is connected, all ICE transports are connected.

ConnectionState **STATE_DISCONNECTED** = `3`

At least one ICE transport is disconnected.

ConnectionState **STATE_FAILED** = `4`

One or more of the ICE transports failed.

ConnectionState **STATE_CLOSED** = `5`

The peer connection is closed (after calling close for example).

enum **GatheringState**: 🔗

GatheringState **GATHERING_STATE_NEW** = `0`

The peer connection was just created and hasn't done any networking yet.

GatheringState **GATHERING_STATE_GATHERING** = `1`

The ICE agent is in the process of gathering candidates for the connection.

GatheringState **GATHERING_STATE_COMPLETE** = `2`

The ICE agent has finished gathering candidates. If something happens that requires collecting new candidates, such as a new interface being added or the addition of a new ICE server, the state will revert to gathering to gather those candidates.

---

enum **SignalingState**: 🔗

SignalingState **SIGNALING_STATE_STABLE** = `0`

There is no ongoing exchange of offer and answer underway. This may mean that the **WebRTCPeerConnection** is new (STATE_NEW) or that negotiation is complete and a connection has been established (STATE_CONNECTED).

SignalingState **SIGNALING_STATE_HAVE_LOCAL_OFFER** = `1`

The local peer has called set_local_description, passing in SDP representing an offer (usually created by calling create_offer), and the offer has been applied successfully.

SignalingState **SIGNALING_STATE_HAVE_REMOTE_OFFER** = `2`

The remote peer has created an offer and used the signaling server to deliver it to the local peer, which has set the offer as the remote description by calling set_remote_description.

SignalingState **SIGNALING_STATE_HAVE_LOCAL_PRANSWER** = `3`

The offer sent by the remote peer has been applied and an answer has been created and applied by calling set_local_description. This provisional answer describes the supported media formats and so forth, but may not have a complete set of ICE candidates included. Further candidates will be delivered separately later.

SignalingState **SIGNALING_STATE_HAVE_REMOTE_PRANSWER** = `4`

A provisional answer has been received and successfully applied in response to an offer previously sent and established by calling set_local_description.

SignalingState **SIGNALING_STATE_CLOSED** = `5`

The **WebRTCPeerConnection** has been closed.

Method Descriptions

Error **add_ice_candidate**(media: String, index: int, name: String)

Add an ice candidate generated by a remote peer (and received over the signaling server). See ice_candidate_created.

void **close**()

Close the peer connection and all data channels associated with it.

Note: You cannot reuse this object for a new connection unless you call initialize.

WebRTCDataChannel **create_data_channel**(label: String, options: Dictionary = {})

Returns a new WebRTCDataChannel (or `null` on failure) with given `label` and optionally configured via the `options` dictionary. This method can only be called when the connection is in state STATE_NEW.

There are two ways to create a working data channel: either call create_data_channel on only one of the peer and listen to data_channel_received on the other, or call create_data_channel on both peers, with the same values, and the `"negotiated"` option set to `true`.

Valid `options` are:

```
{
    "negotiated": true, # When set to true (default off), means
the channel is negotiated out of band. "id" must be set too.
"data_channel_received" will not be called.
    "id": 1, # When "negotiated" is true this value must also
be set to the same value on both peer.

    # Only one of maxRetransmits and maxPacketLifeTime can be
specified, not both. They make the channel unreliable (but also
better at real time).
    "maxRetransmits": 1, # Specify the maximum number of
attempt the peer will make to retransmits packets if they are
not acknowledged.
    "maxPacketLifeTime": 100, # Specify the maximum amount of
time before giving up retransmitions of unacknowledged packets
(in milliseconds).
    "ordered": true, # When in unreliable mode (i.e. either
"maxRetransmits" or "maxPacketLifetime" is set), "ordered"
(true by default) specify if packet ordering is to be enforced.

    "protocol": "my-custom-protocol", # A custom sub-protocol
string for this channel.
}
```

Note: You must keep a reference to channels created this way, or it will be closed.

Error **create_offer**() 🔗

Creates a new SDP offer to start a WebRTC connection with a remote peer. At least one WebRTCDataChannel must have been created before calling this method.

If this functions returns @GlobalScope.OK, session_description_created will be called when the session is ready to be sent.

ConnectionState **get_connection_state**() const

Returns the connection state. See ConnectionState.

GatheringState **get_gathering_state**() const

Returns the ICE GatheringState of the connection. This lets you detect, for example, when collection of ICE candidates has finished.

SignalingState **get_signaling_state**() const

Returns the signaling state on the local end of the connection while connecting or reconnecting to another peer.

Error **initialize**(configuration: Dictionary = {})

Re-initialize this peer connection, closing any previously active connection, and going back to state STATE_NEW. A dictionary of `configuration` options can be passed to configure the peer connection.

Valid `configuration` options are:

```
{
    "iceServers": [
        {
            "urls": [ "stun:stun.example.com:3478" ], # One or more STUN servers.
        },
        {
            "urls": [ "turn:turn.example.com:3478" ], # One or more TURN servers.
            "username": "a_username", # Optional username for the TURN server.
            "credential": "a_password", # Optional password for the TURN server.
        }
    ]
}
```

Error **poll**()

Call this method frequently (e.g. in Node._process or Node._physics_process) to properly receive signals.

void **set_default_extension**(extension_class: StringName) static

Sets the `extension_class` as the default WebRTCPeerConnectionExtension returned when creating a new **WebRTCPeerConnection**.

Error **set_local_description**(type: String, sdp: String)

Sets the SDP description of the local peer. This should be called in response to session_description_created.

After calling this function the peer will start emitting ice_candidate_created (unless an Error different from @GlobalScope.OK is returned).

Error **set_remote_description**(type: String, sdp: String)

Sets the SDP description of the remote peer. This should be called with the values generated by a remote peer and received over the signaling server.

If `type` is `"offer"` the peer will emit session_description_created with the appropriate answer.

If `type` is `"answer"` the peer will start emitting ice_candidate_created.

WebRTCPeerConnectionExtension

Inherits: WebRTCPeerConnection **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

Methods

Error	_add_ice_candidate(p_sdp_mid_name: String, p_sdp_mline_index: int, p_sdp_name: String) virtual
void	_close() virtual
WebRTCDataChannel	_create_data_channel(p_label: String, p_config: Dictionary) virtual
Error	_create_offer() virtual
ConnectionState	_get_connection_state() virtual const
GatheringState	_get_gathering_state() virtual const
SignalingState	_get_signaling_state() virtual const

Error	_initialize(p_config: Dictionary) virtual
Error	_poll() virtual
Error	_set_local_description(p_type: String, p_sdp: String) virtual
Error	_set_remote_description(p_type: String, p_sdp: String) virtual

---

Method Descriptions

Error **_add_ice_candidate**(p_sdp_mid_name: String, p_sdp_mline_index: int, p_sdp_name: String) virtual 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **_close**() virtual 🔗

There is currently no description for this method. Please help us by contributing one!

WebRTCDataChannel **_create_data_channel**(p_label: String, p_config: Dictionary) virtual 🔗

There is currently no description for this method. Please help us by contributing one!

Error **_create_offer**() virtual

There is currently no description for this method. Please help us by contributing one!

ConnectionState **_get_connection_state**() virtual const

There is currently no description for this method. Please help us by contributing one!

GatheringState **_get_gathering_state**() virtual const

There is currently no description for this method. Please help us by contributing one!

SignalingState **_get_signaling_state**() virtual const

There is currently no description for this method. Please help us by contributing one!

Error **_initialize**(p_config: Dictionary) virtual

There is currently no description for this method. Please help us by contributing one!

Error **_poll**() virtual

There is currently no description for this method. Please help us by contributing one!

Error **_set_local_description**(p_type: String, p_sdp: String) virtual 🔗

There is currently no description for this method. Please help us by contributing one!

Error **_set_remote_description**(p_type: String, p_sdp: String) virtual 🔗

There is currently no description for this method. Please help us by contributing one!

WebSocketMultiplayerPeer

Inherits: MultiplayerPeer **<** PacketPeer **<** RefCounted **<** Object

Base class for WebSocket server and client.

Description

Base class for WebSocket server and client, allowing them to be used as multiplayer peer for the MultiplayerAPI.

Note: When exporting to Android, make sure to enable the `INTERNET` permission in the Android export preset before exporting the project or using one-click deploy. Otherwise, network communication of any kind will be blocked by Android.

Properties

PackedStringArray	handshake_headers	`PackedStringArray()`
float	handshake_timeout	`3.0`
int	inbound_buffer_size	`65535`
int	max_queued_packets	`2048`
int	outbound_buffer_size	`65535`

PackedStringArray	supported_protocols	`PackedStringArray()`

Methods

Error	create_client(url: String, tls_client_options: TLSOptions = null)
Error	create_server(port: int, bind_address: String = "*", tls_server_options: TLSOptions = null)
WebSocketPeer	get_peer(peer_id: int) const
String	get_peer_address(id: int) const
int	get_peer_port(id: int) const

Property Descriptions

PackedStringArray **handshake_headers** = `PackedStringArray()` 🔗

- void **set_handshake_headers**(value: PackedStringArray)
- PackedStringArray **get_handshake_headers**()

The extra headers to use during handshake. See WebSocketPeer.handshake_headers for more details.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedStringArray for more details.

float **handshake_timeout** = `3.0`

- void **set_handshake_timeout**(value: float)
- float **get_handshake_timeout**()

The maximum time each peer can stay in a connecting state before being dropped.

int **inbound_buffer_size** = `65535`

- void **set_inbound_buffer_size**(value: int)
- int **get_inbound_buffer_size**()

The inbound buffer size for connected peers. See WebSocketPeer.inbound_buffer_size for more details.

int **max_queued_packets** = `2048`

- void **set_max_queued_packets**(value: int)
- int **get_max_queued_packets**()

The maximum number of queued packets for connected peers. See WebSocketPeer.max_queued_packets for more details.

int **outbound_buffer_size** = `65535`

- void **set_outbound_buffer_size**(value: int)
- int **get_outbound_buffer_size**()

The outbound buffer size for connected peers. See WebSocketPeer.outbound_buffer_size for more details.

PackedStringArray **supported_protocols** = `PackedStringArray()` 🔗

- void **set_supported_protocols**(value: PackedStringArray)
- PackedStringArray **get_supported_protocols**()

The supported WebSocket sub-protocols. See WebSocketPeer.supported_protocols for more details.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedStringArray for more details.

Method Descriptions

Error **create_client**(url: String, tls_client_options: TLSOptions = null) 🔗

Starts a new multiplayer client connecting to the given `url`. TLS certificates will be verified against the hostname when connecting using the `wss://` protocol. You can pass the optional `tls_client_options` parameter to customize the trusted certification authorities, or disable the common name verification. See TLSOptions.client and TLSOptions.client_unsafe.

Note: It is recommended to specify the scheme part of the URL, i.e. the `url` should start with either `ws://` or `wss://`.

Error **create_server**(port: int, bind_address: String = "*", tls_server_options: TLSOptions = null) 🔗

Starts a new multiplayer server listening on the given `port`. You can optionally specify a `bind_address`, and provide valid `tls_server_options` to use TLS. See TLSOptions.server.

WebSocketPeer **get_peer**(peer_id: int) const

Returns the WebSocketPeer associated to the given `peer_id`.

String **get_peer_address**(id: int) const

Returns the IP address of the given peer.

---

int **get_peer_port**(id: int) const

Returns the remote port of the given peer.

WebSocketPeer

Inherits: PacketPeer **<** RefCounted **<** Object

A WebSocket connection.

Description

This class represents WebSocket connection, and can be used as a WebSocket client (RFC 6455-compliant) or as a remote peer of a WebSocket server.

You can send WebSocket binary frames using PacketPeer.put_packet, and WebSocket text frames using send (prefer text frames when interacting with text-based API). You can check the frame type of the last packet via was_string_packet.

To start a WebSocket client, first call connect_to_url, then regularly call poll (e.g. during Node process). You can query the socket state via get_ready_state, get the number of pending packets using PacketPeer.get_available_packet_count, and retrieve them via PacketPeer.get_packet.

GDScript

```
extends Node

var socket = WebSocketPeer.new()

func _ready():
    socket.connect_to_url("wss://example.com")

func _process(delta):
    socket.poll()
    var state = socket.get_ready_state()
    if state == WebSocketPeer.STATE_OPEN:
        while socket.get_available_packet_count():
            print("Packet: ", socket.get_packet())
    elif state == WebSocketPeer.STATE_CLOSING:
        # Keep polling to achieve proper close.
        pass
    elif state == WebSocketPeer.STATE_CLOSED:
        var code = socket.get_close_code()
        var reason = socket.get_close_reason()
        print("WebSocket closed with code: %d, reason %s. Clean: %s" % [code, reason, code != -1])
        set_process(false) # Stop processing.
```

To use the peer as part of a WebSocket server refer to accept_stream and the online tutorial.

Properties

PackedStringArray	handshake_headers	`PackedStringArray()`
int	inbound_buffer_size	`65535`
int	max_queued_packets	`2048`
int	outbound_buffer_size	`65535`
PackedStringArray	supported_protocols	`PackedStringArray()`

Methods

Error	accept_stream(stream: StreamPeer)
void	close(code: int = 1000, reason: String = "")
Error	connect_to_url(url: String, tls_client_options: TLSOptions = null)
int	get_close_code() const
String	get_close_reason() const
String	get_connected_host() const
int	get_connected_port() const
int	get_current_outbound_buffered_amount() const
State	get_ready_state() const
String	get_requested_url() const
String	get_selected_protocol() const
void	poll()

Error	send(message: PackedByteArray, write_mode: WriteMode = 1)
Error	send_text(message: String)
void	set_no_delay(enabled: bool)
bool	was_string_packet() const

Enumerations

enum **WriteMode**: 🔗

WriteMode **WRITE_MODE_TEXT** = `0`

Specifies that WebSockets messages should be transferred as text payload (only valid UTF-8 is allowed).

WriteMode **WRITE_MODE_BINARY** = `1`

Specifies that WebSockets messages should be transferred as binary payload (any byte combination is allowed).

enum **State**: 🔗

State **STATE_CONNECTING** = `0`

Socket has been created. The connection is not yet open.

State **STATE_OPEN** = `1`

The connection is open and ready to communicate.

State **STATE_CLOSING** = `2`

The connection is in the process of closing. This means a close request has been sent to the remote peer but confirmation has not been received.

State **STATE_CLOSED** = `3`

The connection is closed or couldn't be opened.

Property Descriptions

PackedStringArray **handshake_headers** = `PackedStringArray()` 🔗

- void **set_handshake_headers**(value: PackedStringArray)
- PackedStringArray **get_handshake_headers**()

The extra HTTP headers to be sent during the WebSocket handshake.

Note: Not supported in Web exports due to browsers' restrictions.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedStringArray for more details.

---

int **inbound_buffer_size** = `65535` 🔗

- void **set_inbound_buffer_size**(value: int)
- int **get_inbound_buffer_size**()

The size of the input buffer in bytes (roughly the maximum amount of memory that will be allocated for the inbound packets).

---

int **max_queued_packets** = `2048`

- void **set_max_queued_packets**(value: int)
- int **get_max_queued_packets**()

The maximum amount of packets that will be allowed in the queues (both inbound and outbound).

---

int **outbound_buffer_size** = `65535`

- void **set_outbound_buffer_size**(value: int)
- int **get_outbound_buffer_size**()

The size of the input buffer in bytes (roughly the maximum amount of memory that will be allocated for the outbound packets).

PackedStringArray **supported_protocols** = `PackedStringArray()`

- void **set_supported_protocols**(value: PackedStringArray)
- PackedStringArray **get_supported_protocols**()

The WebSocket sub-protocols allowed during the WebSocket handshake.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedStringArray for more details.

Method Descriptions

Error **accept_stream**(stream: StreamPeer) 🔗

Accepts a peer connection performing the HTTP handshake as a WebSocket server. The `stream` must be a valid TCP stream retrieved via TCPServer.take_connection, or a TLS stream accepted via StreamPeerTLS.accept_stream.

Note: Not supported in Web exports due to browsers' restrictions.

---

void **close**(code: int = 1000, reason: String = "") 🔗

Closes this WebSocket connection. `code` is the status code for the closure (see RFC 6455 section 7.4 for a list of valid status codes). `reason` is the human readable reason for closing the connection (can be any UTF-8 string that's smaller than 123 bytes). If `code` is negative, the connection will be closed immediately without notifying the remote peer.

Note: To achieve a clean close, you will need to keep polling until STATE_CLOSED is reached.

Note: The Web export might not support all status codes. Please refer to browser-specific documentation for more details.

Error **connect_to_url**(url: String, tls_client_options: TLSOptions = null) 🔗

Connects to the given URL. TLS certificates will be verified against the hostname when connecting using the `wss://` protocol. You can pass the optional `tls_client_options` parameter to customize the trusted certification authorities, or disable the common name verification. See TLSOptions.client and TLSOptions.client_unsafe.

Note: To avoid mixed content warnings or errors in Web, you may have to use a `url` that starts with `wss://` (secure) instead of `ws://`. When doing so, make sure to use the fully qualified domain name that matches the one defined in the server's TLS certificate. Do not connect directly via the IP address for `wss://` connections, as it won't match with the TLS certificate.

---

int **get_close_code**() const

Returns the received WebSocket close frame status code, or `-1` when the connection was not cleanly closed. Only call this method when get_ready_state returns STATE_CLOSED.

String **get_close_reason**() const

Returns the received WebSocket close frame status reason string. Only call this method when get_ready_state returns STATE_CLOSED.

String **get_connected_host**() const

Returns the IP address of the connected peer.

Note: Not available in the Web export.

---

int **get_connected_port**() const

Returns the remote port of the connected peer.

Note: Not available in the Web export.

---

int **get_current_outbound_buffered_amount**() const

Returns the current amount of data in the outbound websocket buffer. **Note:** Web exports use WebSocket.bufferedAmount, while other platforms use an internal buffer.

State **get_ready_state**() const

Returns the ready state of the connection. See State.

String **get_requested_url**() const

Returns the URL requested by this peer. The URL is derived from the `url` passed to connect_to_url or from the HTTP headers when acting as server (i.e. when using accept_stream).

String **get_selected_protocol**() const

Returns the selected WebSocket sub-protocol for this connection or an empty string if the sub-protocol has not been selected yet.

---

void **poll**()

Updates the connection state and receive incoming packets. Call this function regularly to keep it in a clean state.

Error **send**(message: PackedByteArray, write_mode: WriteMode = 1)

Sends the given `message` using the desired `write_mode`. When sending a String, prefer using send_text.

Error **send_text**(message: String) 🔗

Sends the given `message` using WebSocket text mode. Prefer this method over PacketPeer.put_packet when interacting with third-party text-based API (e.g. when using JSON formatted messages).

---

void **set_no_delay**(enabled: bool) 🔗

Disable Nagle's algorithm on the underlying TCP socket (default). See StreamPeerTCP.set_no_delay for more information.

Note: Not available in the Web export.

---

bool **was_string_packet**() const 🔗

Returns `true` if the last received packet was sent as a text payload. See WriteMode.

WebXRInterface

Inherits: XRInterface **<** RefCounted **<** Object

XR interface using WebXR.

Description

WebXR is an open standard that allows creating VR and AR applications that run in the web browser.

As such, this interface is only available when running in Web exports.

WebXR supports a wide range of devices, from the very capable (like Valve Index, HTC Vive, Oculus Rift and Quest) down to the much less capable (like Google Cardboard, Oculus Go, GearVR, or plain smartphones).

Since WebXR is based on JavaScript, it makes extensive use of callbacks, which means that **WebXRInterface** is forced to use signals, where other XR interfaces would instead use functions that return a result immediately. This makes **WebXRInterface** quite a bit more complicated to initialize than other XR interfaces.

Here's the minimum code required to start an immersive VR session:

```
extends Node3D

var webxr_interface
var vr_supported = false

func _ready():
    # We assume this node has a button as a child.
    # This button is for the user to consent to entering immersive VR
mode.
    $Button.pressed.connect(self._on_button_pressed)

    webxr_interface = XRServer.find_interface("WebXR")
    if webxr_interface:
        # WebXR uses a lot of asynchronous callbacks, so we connect
```

```
to various
        # signals in order to receive them.

webxr_interface.session_supported.connect(self._webxr_session_support
ed)

webxr_interface.session_started.connect(self._webxr_session_started)

webxr_interface.session_ended.connect(self._webxr_session_ended)

webxr_interface.session_failed.connect(self._webxr_session_failed)

        # This returns immediately - our _webxr_session_supported()
method
        # (which we connected to the "session_supported" signal
above) will
        # be called sometime later to let us know if it's supported
or not.
        webxr_interface.is_session_supported("immersive-vr")

func _webxr_session_supported(session_mode, supported):
    if session_mode == 'immersive-vr':
        vr_supported = supported

func _on_button_pressed():
    if not vr_supported:
        OS.alert("Your browser doesn't support VR")
        return

    # We want an immersive VR session, as opposed to AR ('immersive-
ar') or a
    # simple 3DoF viewer ('viewer').
    webxr_interface.session_mode = 'immersive-vr'
    # 'bounded-floor' is room scale, 'local-floor' is a standing or
sitting
    # experience (it puts you 1.6m above the ground if you have 3DoF
headset),
    # whereas as 'local' puts you down at the XROrigin.
    # This list means it'll first try to request 'bounded-floor',
then
    # fallback on 'local-floor' and ultimately 'local', if nothing
else is
    # supported.
    webxr_interface.requested_reference_space_types = 'bounded-floor,
local-floor, local'
    # In order to use 'local-floor' or 'bounded-floor' we must also
    # mark the features as required or optional. By including 'hand-
tracking'
    # as an optional feature, it will be enabled if supported.
    webxr_interface.required_features = 'local-floor'
    webxr_interface.optional_features = 'bounded-floor, hand-
tracking'

    # This will return false if we're unable to even request the
session,
    # however, it can still fail asynchronously later in the process,
so we
    # only know if it's really succeeded or failed when our
    # _webxr_session_started() or _webxr_session_failed() methods are
called.
    if not webxr_interface.initialize():
        OS.alert("Failed to initialize")
        return

func _webxr_session_started():
    $Button.visible = false
    # This tells Godot to start rendering to the headset.
    get_viewport().use_xr = true
    # This will be the reference space type you ultimately got, out
of the
    # types that you requested above. This is useful if you want the
game to
    # work a little differently in 'bounded-floor' versus 'local-
floor'.
    print("Reference space type: ",
webxr_interface.reference_space_type)
    # This will be the list of features that were successfully
enabled
    # (except on browsers that don't support this property).
    print("Enabled features: ", webxr_interface.enabled_features)

func _webxr_session_ended():
    $Button.visible = true
    # If the user exits immersive mode, then we tell Godot to render
to the web
    # page again.
    get_viewport().use_xr = false

func _webxr_session_failed(message):
    OS.alert("Failed to initialize: " + message)
```

There are a couple ways to handle "controller" input:

- Using XRController3D nodes and their XRController3D.button_pressed and XRController3D.button_released signals. This is how controllers are typically handled in XR apps in Godot, however, this will only work with advanced VR controllers like the Oculus Touch or Index controllers, for example.
- Using the select, squeeze and related signals. This method will work for both advanced VR controllers, and non-traditional input sources like a tap on the screen, a spoken voice command or a button press on the device itself.

You can use both methods to allow your game or app to support a wider or narrower set of devices and input methods, or to allow more advanced interactions with more advanced devices.

Tutorials

- How to make a VR game for WebXR with Godot 4 [https://www.snopekgames.com/tutorial/2023/how-make-vr-game-webxr-godot-4]

Properties

String	enabled_features
String	optional_features
String	reference_space_type
String	requested_reference_space_types
String	required_features
String	session_mode

String	visibility_state

Methods

Array	get_available_display_refresh_rates() const
float	get_display_refresh_rate() const
TargetRayMode	get_input_source_target_ray_mode(input_source_id: int) const
XRControllerTracker	get_input_source_tracker(input_source_id: int) const
bool	is_input_source_active(input_source_id: int) const
void	is_session_supported(session_mode: String)
void	set_display_refresh_rate(refresh_rate: float)

Signals

display_refresh_rate_changed() 🔗

Emitted after the display's refresh rate has changed.

---

reference_space_reset() 🔗

Emitted to indicate that the reference space has been reset or reconfigured.

When (or whether) this is emitted depends on the user's browser or device, but may include when the user has changed the dimensions of their play space (which you may be able to access via XRInterface.get_play_area) or pressed/held a button to recenter their position.

See WebXR's XRReferenceSpace reset event [https://developer.mozilla.org/en-US/docs/Web/API/XRReferenceSpace/reset_event] for more information.

---

select(input_source_id: int) 🔗

Emitted after one of the input sources has finished its "primary action".

Use get_input_source_tracker and get_input_source_target_ray_mode to get more information about the input source.

---

selectend(input_source_id: int) 🔗

Emitted when one of the input sources has finished its "primary action".

Use get_input_source_tracker and get_input_source_target_ray_mode to get more information about the input source.

---

selectstart(input_source_id: int) 🔗

Emitted when one of the input source has started its "primary action".

Use get_input_source_tracker and get_input_source_target_ray_mode to get more information about the input source.

---

session_ended() 🔗

Emitted when the user ends the WebXR session (which can be done using UI from the browser or device).

At this point, you should do `get_viewport().use_xr = false` to instruct Godot to resume rendering to the screen.

---

session_failed(message: String)

Emitted by XRInterface.initialize if the session fails to start.

`message` may optionally contain an error message from WebXR, or an empty string if no message is available.

---

session_started()

Emitted by XRInterface.initialize if the session is successfully started.

At this point, it's safe to do `get_viewport().use_xr = true` to instruct Godot to start rendering to the XR device.

---

session_supported(session_mode: String, supported: bool)

Emitted by is_session_supported to indicate if the given `session_mode` is supported or not.

---

squeeze(input_source_id: int)

Emitted after one of the input sources has finished its "primary squeeze action".

Use get_input_source_tracker and get_input_source_target_ray_mode to get more information about the input source.

---

squeezeend(input_source_id: int)

Emitted when one of the input sources has finished its "primary squeeze action".

Use get_input_source_tracker and get_input_source_target_ray_mode to get more information about the input source.

---

squeezestart(input_source_id: int)

Emitted when one of the input sources has started its "primary squeeze action".

Use get_input_source_tracker and get_input_source_target_ray_mode to get more information about the input source.

---

visibility_state_changed()

Emitted when visibility_state has changed.

Enumerations

enum **TargetRayMode**:

TargetRayMode **TARGET_RAY_MODE_UNKNOWN** = `0`

We don't know the the target ray mode.

TargetRayMode **TARGET_RAY_MODE_GAZE** = `1`

Target ray originates at the viewer's eyes and points in the direction they are looking.

TargetRayMode **TARGET_RAY_MODE_TRACKED_POINTER** = `2`

Target ray from a handheld pointer, most likely a VR touch controller.

TargetRayMode **TARGET_RAY_MODE_SCREEN** = `3`

Target ray from touch screen, mouse or other tactile input device.

Property Descriptions

String **enabled_features** 🔗

- String **get_enabled_features**()

A comma-separated list of features that were successfully enabled by XRInterface.initialize when setting up the WebXR session.

This may include features requested by setting required_features and optional_features, and will only be available after session_started has been emitted.

Note: This may not be support by all web browsers, in which case it will be an empty string.

String **optional_features** 🔗

- void **set_optional_features**(value: String)
- String **get_optional_features**()

A comma-seperated list of optional features used by XRInterface.initialize when setting up the WebXR session.

If a user's browser or device doesn't support one of the given features, initialization will continue, but you won't be able to use the requested feature.

This doesn't have any effect on the interface when already initialized.

Possible values come from WebXR's XRReferenceSpaceType [https://developer.mozilla.org/en-US/docs/Web/API/XRReferenceSpaceType], or include other features like `"hand-tracking"` to enable hand tracking.

String **reference_space_type**

- String **get_reference_space_type**()

The reference space type (from the list of requested types set in the requested_reference_space_types property), that was ultimately used by XRInterface.initialize when setting up the WebXR session.

Possible values come from WebXR's XRReferenceSpaceType [https://developer.mozilla.org/en-US/docs/Web/API/XRReferenceSpaceType]. If you want to use a particular reference space type, it must be listed in either required_features or optional_features.

String **requested_reference_space_types**

- void **set_requested_reference_space_types**(value: String)
- String **get_requested_reference_space_types**()

A comma-seperated list of reference space types used by XRInterface.initialize when setting up the WebXR session.

The reference space types are requested in order, and the first one supported by the users device or browser will be used. The reference_space_type property contains the reference space type that was ultimately selected.

This doesn't have any effect on the interface when already initialized.

Possible values come from WebXR's XRReferenceSpaceType [https://developer.mozilla.org/en-US/docs/Web/API/XRReferenceSpaceType]. If you want to use a particular reference space type, it must be listed in either required_features or optional_features.

String **required_features**

- void **set_required_features**(value: String)
- String **get_required_features**()

A comma-seperated list of required features used by XRInterface.initialize when setting up the WebXR session.

If a user's browser or device doesn't support one of the given features, initialization will fail and session_failed will be emitted.

This doesn't have any effect on the interface when already initialized.

Possible values come from WebXR's XRReferenceSpaceType [https://developer.mozilla.org/en-US/docs/Web/API/XRReferenceSpaceType], or include other features like `"hand-tracking"` to enable hand tracking.

String **session_mode** 🔗

- void **set_session_mode**(value: String)
- String **get_session_mode**()

The session mode used by XRInterface.initialize when setting up the WebXR session.

This doesn't have any effect on the interface when already initialized.

Possible values come from WebXR's XRSessionMode [https://developer.mozilla.org/en-US/docs/Web/API/XRSessionMode], including: `"immersive-vr"`, `"immersive-ar"`, and `"inline"`.

String **visibility_state** 🔗

- String **get_visibility_state**()

Indicates if the WebXR session's imagery is visible to the user.

Possible values come from WebXR's XRVisibilityState [https://developer.mozilla.org/en-US/docs/Web/API/XRVisibilityState], including `"hidden"`, `"visible"`, and `"visible-blurred"`.

Method Descriptions

Array **get_available_display_refresh_rates**() const

Returns display refresh rates supported by the current HMD. Only returned if this feature is supported by the web browser and after the interface has been initialized.

---

float **get_display_refresh_rate**() const

Returns the display refresh rate for the current HMD. Not supported on all HMDs and browsers. It may not report an accurate value until after using set_display_refresh_rate.

TargetRayMode **get_input_source_target_ray_mode**(input_source_id: int) const

Returns the target ray mode for the given `input_source_id`.

This can help interpret the input coming from that input source. See XRInputSource.targetRayMode [https://developer.mozilla.org/en-US/docs/Web/API/XRInputSource/targetRayMode] for more information.

XRControllerTracker **get_input_source_tracker**(input_source_id: int) const

Gets an XRControllerTracker for the given `input_source_id`.

In the context of WebXR, an input source can be an advanced VR controller like the Oculus Touch or Index controllers, or even a tap on the screen, a spoken voice command or a button press on the device itself. When a non-traditional input source is used, interpret the position and orientation of the XRPositionalTracker as a ray pointing at the object the user wishes to interact with.

Use this method to get information about the input source that triggered one of these signals:

- selectstart

- select
- selectend
- squeezestart
- squeeze
- squeezestart

---

bool **is_input_source_active**(input_source_id: int) const 🔗

Returns `true` if there is an active input source with the given `input_source_id`.

---

void **is_session_supported**(session_mode: String) 🔗

Checks if the given `session_mode` is supported by the user's browser.

Possible values come from WebXR's XRSessionMode [https://developer.mozilla.org/en-US/docs/Web/API/XRSessionMode], including: `"immersive-vr"`, `"immersive-ar"`, and `"inline"`.

This method returns nothing, instead it emits the session_supported signal with the result.

---

void **set_display_refresh_rate**(refresh_rate: float) 🔗

Sets the display refresh rate for the current HMD. Not supported on all HMDs and browsers. It won't take effect right away until after display_refresh_rate_changed is emitted.

WorkerThreadPool

Inherits: Object

A singleton that allocates some Threads on startup, used to offload tasks to these threads.

Description

The **WorkerThreadPool** singleton allocates a set of Threads (called worker threads) on project startup and provides methods for offloading tasks to them. This can be used for simple multithreading without having to create Threads.

Tasks hold the Callable to be run by the threads. **WorkerThreadPool** can be used to create regular tasks, which will be taken by one worker thread, or group tasks, which can be distributed between multiple worker threads. Group tasks execute the Callable multiple times, which makes them useful for iterating over a lot of elements, such as the enemies in an arena.

Here's a sample on how to offload an expensive function to worker threads:

GDScript

```
var enemies = [] # An array to be filled with enemies.

func process_enemy_ai(enemy_index):
    var processed_enemy = enemies[enemy_index]
    # Expensive logic...

func _process(delta):
    var task_id =
WorkerThreadPool.add_group_task(process_enemy_ai,
enemies.size())
    # Other code...
    WorkerThreadPool.wait_for_group_task_completion(task_id)
    # Other code that depends on the enemy AI already being
processed.
```

C#

```
private List<Node> _enemies = new List<Node>(); // A list to be
filled with enemies.

private void ProcessEnemyAI(int enemyIndex)
{
    Node processedEnemy = _enemies[enemyIndex];
    // Expensive logic here.
}

public override void _Process(double delta)
{
    long taskId =
WorkerThreadPool.AddGroupTask(Callable.From<int>
(ProcessEnemyAI), _enemies.Count);
    // Other code...
    WorkerThreadPool.WaitForGroupTaskCompletion(taskId);
    // Other code that depends on the enemy AI already being
processed.
}
```

The above code relies on the number of elements in the `enemies` array remaining constant during the multithreaded part.

Note: Using this singleton could affect performance negatively if the task being distributed between threads is not computationally expensive.

Tutorials

- Using multiple threads

- Thread-safe APIs

Methods

int	add_group_task(action: Callable, elements: int, tasks_needed: int = -1, high_priority: bool = false, description: String = "")
int	add_task(action: Callable, high_priority: bool = false, description: String = "")
int	get_group_processed_element_count(group_id: int) const
bool	is_group_task_completed(group_id: int) const
bool	is_task_completed(task_id: int) const
void	wait_for_group_task_completion(group_id: int)
Error	wait_for_task_completion(task_id: int)

Method Descriptions

int **add_group_task**(action: Callable, elements: int, tasks_needed: int = -1, high_priority: bool = false, description: String = "") 🔗

Adds `action` as a group task to be executed by the worker threads. The Callable will be called a number of times based on `elements`, with the first thread calling it with the value `0` as a parameter, and each consecutive execution incrementing this value by 1 until it reaches `element - 1`.

The number of threads the task is distributed to is defined by `tasks_needed`, where the default value `-1` means it is distributed to all worker threads. `high_priority` determines if the task has a high priority or a low priority (default). You can optionally provide a `description` to help with debugging.

Returns a group task ID that can be used by other methods.

Warning: Every task must be waited for completion using wait_for_task_completion or wait_for_group_task_completion at some point so that any allocated resources inside the task can be cleaned up.

---

int **add_task**(action: Callable, high_priority: bool = false, description: String = "") 🔗

Adds `action` as a task to be executed by a worker thread. `high_priority` determines if the task has a high priority or a low priority (default). You can optionally provide a `description` to help with debugging.

Returns a task ID that can be used by other methods.

Warning: Every task must be waited for completion using wait_for_task_completion or wait_for_group_task_completion at some point so that any allocated resources inside the task can be cleaned up.

---

int **get_group_processed_element_count**(group_id: int) const

Returns how many times the Callable of the group task with the given ID has already been executed by the worker threads.

Note: If a thread has started executing the Callable but is yet to finish, it won't be counted.

---

bool **is_group_task_completed**(group_id: int) const

Returns `true` if the group task with the given ID is completed.

Note: You should only call this method between adding the group task and awaiting its completion.

---

bool **is_task_completed**(task_id: int) const

Returns `true` if the task with the given ID is completed.

Note: You should only call this method between adding the task and awaiting its completion.

---

void **wait_for_group_task_completion**(group_id: int)

Pauses the thread that calls this method until the group task with the given ID is completed.

Error **wait_for_task_completion**(task_id: int)

Pauses the thread that calls this method until the task with the given ID is completed.

Returns @GlobalScope.OK if the task could be successfully awaited.

Returns @GlobalScope.ERR_INVALID_PARAMETER if a task with the passed ID does not exist (maybe because it was already awaited and disposed of).

Returns @GlobalScope.ERR_BUSY if the call is made from another running task and, due to task scheduling, there's potential for deadlocking (e.g., the task to await may be at a lower level in the call stack and therefore can't progress). This is an advanced situation that should only matter when some tasks depend on others (in the current implementation, the tricky case is a task trying to wait on an older one).

XMLParser

Inherits: RefCounted **<** Object

Provides a low-level interface for creating parsers for XML files.

Description

Provides a low-level interface for creating parsers for XML [https://en.wikipedia.org/wiki/XML] files. This class can serve as base to make custom XML parsers.

To parse XML, you must open a file with the open method or a buffer with the open_buffer method. Then, the read method must be called to parse the next nodes. Most of the methods take into consideration the currently parsed node.

Here is an example of using **XMLParser** to parse an SVG file (which is based on XML), printing each element and its attributes as a dictionary:

GDScript

```
var parser = XMLParser.new()
parser.open("path/to/file.svg")
while parser.read() != ERR_FILE_EOF:
    if parser.get_node_type() == XMLParser.NODE_ELEMENT:
        var node_name = parser.get_node_name()
        var attributes_dict = {}
        for idx in range(parser.get_attribute_count()):
            attributes_dict[parser.get_attribute_name(idx)] = parser.get_attribute_value(idx)
        print("The ", node_name, " element has the following attributes: ", attributes_dict)
```

C#

```
var parser = new XmlParser();
parser.Open("path/to/file.svg");
while (parser.Read() != Error.FileEof)
{
    if (parser.GetNodeType() == XmlParser.NodeType.Element)
    {
        var nodeName = parser.GetNodeName();
        var attributesDict = new Godot.Collections.Dictionary();
        for (int idx = 0; idx < parser.GetAttributeCount(); idx++)
        {
            attributesDict[parser.GetAttributeName(idx)] = parser.GetAttributeValue(idx);
        }
        GD.Print($"The {nodeName} element has the following attributes: {attributesDict}");
    }
}
```

Methods

int	get_attribute_count() const
String	get_attribute_name(idx: int) const
String	get_attribute_value(idx: int) const
int	get_current_line() const
String	get_named_attribute_value(name: String) const

String	get_named_attribute_value_safe(name: String) const
String	get_node_data() const
String	get_node_name() const
int	get_node_offset() const
NodeType	get_node_type()
bool	has_attribute(name: String) const
bool	is_empty() const
Error	open(file: String)
Error	open_buffer(buffer: PackedByteArray)
Error	read()
Error	seek(position: int)
void	skip_section()

Enumerations

enum **NodeType**: 🔗

NodeType **NODE_NONE** = `0`

There's no node (no file or buffer opened).

NodeType **NODE_ELEMENT** = `1`

An element node type, also known as a tag, e.g. `<title>`.

NodeType **NODE_ELEMENT_END** = `2`

An end of element node type, e.g. `</title>`.

NodeType **NODE_TEXT** = `3`

A text node type, i.e. text that is not inside an element. This includes whitespace.

NodeType **NODE_COMMENT** = `4`

A comment node type, e.g. `<!--A comment-->`.

NodeType **NODE_CDATA** = `5`

A node type for CDATA (Character Data) sections, e.g. `<![CDATA[CDATA section]]>`.

NodeType **NODE_UNKNOWN** = `6`

An unknown node type.

Method Descriptions

int **get_attribute_count**() const

Returns the number of attributes in the currently parsed element.

Note: If this method is used while the currently parsed node is not NODE_ELEMENT or NODE_ELEMENT_END, this count will not be updated and will still reflect the last element.

String **get_attribute_name**(idx: int) const

Returns the name of an attribute of the currently parsed element, specified by the `idx` index.

String **get_attribute_value**(idx: int) const

Returns the value of an attribute of the currently parsed element, specified by the `idx` index.

---

int **get_current_line**() const

Returns the current line in the parsed file, counting from 0.

String **get_named_attribute_value**(name: String) const

Returns the value of an attribute of the currently parsed element, specified by its `name`. This method will raise an error if the element has no such attribute.

String **get_named_attribute_value_safe**(name: String) const

Returns the value of an attribute of the currently parsed element, specified by its `name`. This method will return an empty string if the element has no such attribute.

String **get_node_data**() const

Returns the contents of a text node. This method will raise an error if the current parsed node is of any other type.

String **get_node_name**() const

Returns the name of a node. This method will raise an error if the currently parsed node is a text node.

Note: The content of a NODE_CDATA node and the comment string of a NODE_COMMENT node are also considered names.

---

int **get_node_offset**() const

Returns the byte offset of the currently parsed node since the beginning of the file or buffer. This is usually equivalent to the number of characters before the read position.

NodeType **get_node_type**()

Returns the type of the current node. Compare with NodeType constants.

---

bool **has_attribute**(name: String) const

Returns `true` if the currently parsed element has an attribute with the `name`.

---

bool **is_empty**() const 🔗

Returns `true` if the currently parsed element is empty, e.g. `<element />`.

Error **open**(file: String) 🔗

Opens an XML `file` for parsing. This method returns an error code.

Error **open_buffer**(buffer: PackedByteArray) 🔗

Opens an XML raw `buffer` for parsing. This method returns an error code.

Error **read**() 🔗

Parses the next node in the file. This method returns an error code.

Error **seek**(position: int) 🔗

Moves the buffer cursor to a certain offset (since the beginning) and reads the next node there. This method returns an error code.

---

void **skip_section**() 🔗

Skips the current section. If the currently parsed node contains more inner nodes, they will be ignored and the cursor will go to the closing of the current element.

XRBodyTracker

Experimental: This class may be changed or removed in future versions.

Inherits: XRPositionalTracker **<** XRTracker **<** RefCounted **<** Object

A tracked body in XR.

Description

A body tracking system will create an instance of this object and add it to the XRServer. This tracking system will then obtain skeleton data, convert it to the Godot Humanoid skeleton and store this data on the **XRBodyTracker** object.

Use XRBodyModifier3D to animate a body mesh using body tracking data.

Tutorials

- XR documentation index

Properties

BitField[BodyFlags]	body_flags	`0`
bool	has_tracking_data	`false`

TrackerType	type	`32` (overrides XRTracker)

Methods

BitField[JointFlags]	get_joint_flags(joint: Joint) const
Transform3D	get_joint_transform(joint: Joint) const
void	set_joint_flags(joint: Joint, flags: BitField[JointFlags])
void	set_joint_transform(joint: Joint, transform: Transform3D)

Enumerations

flags **BodyFlags**: 🔗

BodyFlags **BODY_FLAG_UPPER_BODY_SUPPORTED** = `1`

Upper body tracking supported.

BodyFlags **BODY_FLAG_LOWER_BODY_SUPPORTED** = `2`

Lower body tracking supported.

BodyFlags **BODY_FLAG_HANDS_SUPPORTED** = `4`

Hand tracking supported.

---

enum **Joint**: 🔗

Joint **JOINT_ROOT** = `0`

Root joint.

Joint **JOINT_HIPS** = `1`

Hips joint.

Joint **JOINT_SPINE** = `2`

Spine joint.

Joint **JOINT_CHEST** = `3`

Chest joint.

Joint **JOINT_UPPER_CHEST** = `4`

Upper chest joint.

Joint **JOINT_NECK** = `5`

Neck joint.

Joint **JOINT_HEAD** = `6`

Head joint.

Joint **JOINT_HEAD_TIP** = `7`

Head tip joint.

Joint **JOINT_LEFT_SHOULDER** = `8`

Left shoulder joint.

Joint **JOINT_LEFT_UPPER_ARM** = `9`

Left upper arm joint.

Joint **JOINT_LEFT_LOWER_ARM** = `10`

Left lower arm joint.

Joint **JOINT_RIGHT_SHOULDER** = `11`

Right shoulder joint.

Joint **JOINT_RIGHT_UPPER_ARM** = `12`

Right upper arm joint.

Joint **JOINT_RIGHT_LOWER_ARM** = `13`

Right lower arm joint.

Joint **JOINT_LEFT_UPPER_LEG** = `14`

Left upper leg joint.

Joint **JOINT_LEFT_LOWER_LEG** = `15`

Left lower leg joint.

Joint **JOINT_LEFT_FOOT** = `16`

Left foot joint.

Joint **JOINT_LEFT_TOES** = `17`

Left toes joint.

Joint **JOINT_RIGHT_UPPER_LEG** = `18`

Right upper leg joint.

Joint **JOINT_RIGHT_LOWER_LEG** = `19`

Right lower leg joint.

Joint **JOINT_RIGHT_FOOT** = `20`

Right foot joint.

Joint **JOINT_RIGHT_TOES** = `21`

Right toes joint.

Joint **JOINT_LEFT_HAND** = `22`

Left hand joint.

Joint **JOINT_LEFT_PALM** = `23`

Left palm joint.

Joint **JOINT_LEFT_WRIST** = `24`

Left wrist joint.

Joint **JOINT_LEFT_THUMB_METACARPAL** = `25`

Left thumb metacarpal joint.

Joint **JOINT_LEFT_THUMB_PHALANX_PROXIMAL** = `26`

Left thumb phalanx proximal joint.

Joint **JOINT_LEFT_THUMB_PHALANX_DISTAL** = `27`

Left thumb phalanx distal joint.

Joint **JOINT_LEFT_THUMB_TIP** = `28`

Left thumb tip joint.

Joint **JOINT_LEFT_INDEX_FINGER_METACARPAL** = `29`

Left index finger metacarpal joint.

Joint **JOINT_LEFT_INDEX_FINGER_PHALANX_PROXIMAL** = `30`

Left index finger phalanx proximal joint.

Joint **JOINT_LEFT_INDEX_FINGER_PHALANX_INTERMEDIATE** = `31`

Left index finger phalanx intermediate joint.

Joint **JOINT_LEFT_INDEX_FINGER_PHALANX_DISTAL** = `32`

Left index finger phalanx distal joint.

Joint **JOINT_LEFT_INDEX_FINGER_TIP** = `33`

Left index finger tip joint.

Joint **JOINT_LEFT_MIDDLE_FINGER_METACARPAL** = `34`

Left middle finger metacarpal joint.

Joint **JOINT_LEFT_MIDDLE_FINGER_PHALANX_PROXIMAL** = `35`

Left middle finger phalanx proximal joint.

Joint **JOINT_LEFT_MIDDLE_FINGER_PHALANX_INTERMEDIATE** = `36`

Left middle finger phalanx intermediate joint.

Joint **JOINT_LEFT_MIDDLE_FINGER_PHALANX_DISTAL** = `37`

Left middle finger phalanx distal joint.

Joint **JOINT_LEFT_MIDDLE_FINGER_TIP** = `38`

Left middle finger tip joint.

Joint **JOINT_LEFT_RING_FINGER_METACARPAL** = `39`

Left ring finger metacarpal joint.

Joint **JOINT_LEFT_RING_FINGER_PHALANX_PROXIMAL** = `40`

Left ring finger phalanx proximal joint.

Joint **JOINT_LEFT_RING_FINGER_PHALANX_INTERMEDIATE** = `41`

Left ring finger phalanx intermediate joint.

Joint **JOINT_LEFT_RING_FINGER_PHALANX_DISTAL** = `42`

Left ring finger phalanx distal joint.

Joint **JOINT_LEFT_RING_FINGER_TIP** = `43`

Left ring finger tip joint.

Joint **JOINT_LEFT_PINKY_FINGER_METACARPAL** = `44`

Left pinky finger metacarpal joint.

Joint **JOINT_LEFT_PINKY_FINGER_PHALANX_PROXIMAL** = `45`

Left pinky finger phalanx proximal joint.

Joint **JOINT_LEFT_PINKY_FINGER_PHALANX_INTERMEDIATE** = `46`

Left pinky finger phalanx intermediate joint.

Joint **JOINT_LEFT_PINKY_FINGER_PHALANX_DISTAL** = `47`

Left pinky finger phalanx distal joint.

Joint **JOINT_LEFT_PINKY_FINGER_TIP** = `48`

Left pinky finger tip joint.

Joint **JOINT_RIGHT_HAND** = `49`

Right hand joint.

Joint **JOINT_RIGHT_PALM** = `50`

Right palm joint.

Joint **JOINT_RIGHT_WRIST** = `51`

Right wrist joint.

Joint **JOINT_RIGHT_THUMB_METACARPAL** = `52`

Right thumb metacarpal joint.

Joint **JOINT_RIGHT_THUMB_PHALANX_PROXIMAL** = `53`

Right thumb phalanx proximal joint.

Joint **JOINT_RIGHT_THUMB_PHALANX_DISTAL** = `54`

Right thumb phalanx distal joint.

Joint **JOINT_RIGHT_THUMB_TIP** = `55`

Right thumb tip joint.

Joint **JOINT_RIGHT_INDEX_FINGER_METACARPAL** = `56`

Right index finger metacarpal joint.

Joint **JOINT_RIGHT_INDEX_FINGER_PHALANX_PROXIMAL** = `57`

Right index finger phalanx proximal joint.

Joint **JOINT_RIGHT_INDEX_FINGER_PHALANX_INTERMEDIATE** = `58`

Right index finger phalanx intermediate joint.

Joint **JOINT_RIGHT_INDEX_FINGER_PHALANX_DISTAL** = `59`

Right index finger phalanx distal joint.

Joint **JOINT_RIGHT_INDEX_FINGER_TIP** = `60`

Right index finger tip joint.

Joint **JOINT_RIGHT_MIDDLE_FINGER_METACARPAL** = `61`

Right middle finger metacarpal joint.

Joint **JOINT_RIGHT_MIDDLE_FINGER_PHALANX_PROXIMAL** = `62`

Right middle finger phalanx proximal joint.

Joint **JOINT_RIGHT_MIDDLE_FINGER_PHALANX_INTERMEDIATE** = `63`

Right middle finger phalanx intermediate joint.

Joint **JOINT_RIGHT_MIDDLE_FINGER_PHALANX_DISTAL** = `64`

Right middle finger phalanx distal joint.

Joint **JOINT_RIGHT_MIDDLE_FINGER_TIP** = `65`

Right middle finger tip joint.

Joint **JOINT_RIGHT_RING_FINGER_METACARPAL** = `66`

Right ring finger metacarpal joint.

Joint **JOINT_RIGHT_RING_FINGER_PHALANX_PROXIMAL** = `67`

Right ring finger phalanx proximal joint.

Joint **JOINT_RIGHT_RING_FINGER_PHALANX_INTERMEDIATE** = `68`

Right ring finger phalanx intermediate joint.

Joint **JOINT_RIGHT_RING_FINGER_PHALANX_DISTAL** = `69`

Right ring finger phalanx distal joint.

Joint **JOINT_RIGHT_RING_FINGER_TIP** = `70`

Right ring finger tip joint.

Joint **JOINT_RIGHT_PINKY_FINGER_METACARPAL** = `71`

Right pinky finger metacarpal joint.

Joint **JOINT_RIGHT_PINKY_FINGER_PHALANX_PROXIMAL** = `72`

Right pinky finger phalanx proximal joint.

Joint **JOINT_RIGHT_PINKY_FINGER_PHALANX_INTERMEDIATE** = `73`

Right pinky finger phalanx intermediate joint.

Joint **JOINT_RIGHT_PINKY_FINGER_PHALANX_DISTAL** = `74`

Right pinky finger phalanx distal joint.

Joint **JOINT_RIGHT_PINKY_FINGER_TIP** = `75`

Right pinky finger tip joint.

Joint **JOINT_MAX** = `76`

Represents the size of the Joint enum.

flags **JointFlags**: 🔗

JointFlags **JOINT_FLAG_ORIENTATION_VALID** = `1`

The joint's orientation data is valid.

JointFlags **JOINT_FLAG_ORIENTATION_TRACKED** = `2`

The joint's orientation is actively tracked. May not be set if tracking has been temporarily lost.

JointFlags **JOINT_FLAG_POSITION_VALID** = `4`

The joint's position data is valid.

JointFlags **JOINT_FLAG_POSITION_TRACKED** = `8`

The joint's position is actively tracked. May not be set if tracking has been temporarily lost.

Property Descriptions

BitField[BodyFlags] **body_flags** = `0` 🔗

- void **set_body_flags**(value: BitField[BodyFlags])
- BitField[BodyFlags] **get_body_flags**()

The type of body tracking data captured.

---

bool **has_tracking_data** = `false` 🔗

- void **set_has_tracking_data**(value: bool)
- bool **get_has_tracking_data**()

If `true`, the body tracking data is valid.

Method Descriptions

BitField[JointFlags] **get_joint_flags**(joint: Joint) const 🔗

Returns flags about the validity of the tracking data for the given body joint (see JointFlags).

Transform3D **get_joint_transform**(joint: Joint) const

Returns the transform for the given body joint.

---

void **set_joint_flags**(joint: Joint, flags: BitField[JointFlags])

Sets flags about the validity of the tracking data for the given body joint.

---

void **set_joint_transform**(joint: Joint, transform: Transform3D)

Sets the transform for the given body joint.

XRControllerTracker

Inherits: XRPositionalTracker **<** XRTracker **<** RefCounted **<** Object

A tracked controller.

Description

An instance of this object represents a controller that is tracked.

As controllers are turned on and the XRInterface detects them, instances of this object are automatically added to this list of active tracking objects accessible through the XRServer.

The XRController3D consumes objects of this type and should be used in your project.

Tutorials

- XR documentation index

Properties

TrackerType	type	`2` (overrides XRTracker)

XRFaceTracker

Experimental: This class may be changed or removed in future versions.

Inherits: XRTracker **<** RefCounted **<** Object

A tracked face.

Description

An instance of this object represents a tracked face and its corresponding blend shapes. The blend shapes come from the Unified Expressions [https://docs.vrcft.io/docs/tutorial-avatars/tutorial-avatars-extras/unified-blendshapes] standard, and contain extended details and visuals for each blend shape. Additionally the Tracking Standard Comparison [https://docs.vrcft.io/docs/tutorial-avatars/tutorial-avatars-extras/compatibility/overview] page documents the relationship between Unified Expressions and other standards.

As face trackers are turned on they are registered with the XRServer.

Tutorials

- XR documentation index

Properties

PackedFloat32Array	blend_shapes	`PackedFloat32Array()`

TrackerType	type	`64` (overrides XRTracker)

Methods

float	get_blend_shape(blend_shape: BlendShapeEntry) const
void	set_blend_shape(blend_shape: BlendShapeEntry, weight: float)

Enumerations

enum **BlendShapeEntry**: 🔗

BlendShapeEntry **FT_EYE_LOOK_OUT_RIGHT** = `0`

Right eye looks outwards.

BlendShapeEntry **FT_EYE_LOOK_IN_RIGHT** = `1`

Right eye looks inwards.

BlendShapeEntry **FT_EYE_LOOK_UP_RIGHT** = `2`

Right eye looks upwards.

BlendShapeEntry **FT_EYE_LOOK_DOWN_RIGHT** = `3`

Right eye looks downwards.

BlendShapeEntry **FT_EYE_LOOK_OUT_LEFT** = `4`

Left eye looks outwards.

BlendShapeEntry **FT_EYE_LOOK_IN_LEFT** = `5`

Left eye looks inwards.

BlendShapeEntry **FT_EYE_LOOK_UP_LEFT** = `6`

Left eye looks upwards.

BlendShapeEntry **FT_EYE_LOOK_DOWN_LEFT** = `7`

Left eye looks downwards.

BlendShapeEntry **FT_EYE_CLOSED_RIGHT** = `8`

Closes the right eyelid.

BlendShapeEntry **FT_EYE_CLOSED_LEFT** = `9`

Closes the left eyelid.

BlendShapeEntry **FT_EYE_SQUINT_RIGHT** = `10`

Squeezes the right eye socket muscles.

BlendShapeEntry **FT_EYE_SQUINT_LEFT** = `11`

Squeezes the left eye socket muscles.

BlendShapeEntry **FT_EYE_WIDE_RIGHT** = `12`

Right eyelid widens beyond relaxed.

BlendShapeEntry **FT_EYE_WIDE_LEFT** = `13`

Left eyelid widens beyond relaxed.

BlendShapeEntry **FT_EYE_DILATION_RIGHT** = `14`

Dilates the right eye pupil.

BlendShapeEntry **FT_EYE_DILATION_LEFT** = `15`

Dilates the left eye pupil.

BlendShapeEntry **FT_EYE_CONSTRICT_RIGHT** = `16`

Constricts the right eye pupil.

BlendShapeEntry **FT_EYE_CONSTRICT_LEFT** = `17`

Constricts the left eye pupil.

BlendShapeEntry **FT_BROW_PINCH_RIGHT** = `18`

Right eyebrow pinches in.

BlendShapeEntry **FT_BROW_PINCH_LEFT** = `19`

Left eyebrow pinches in.

BlendShapeEntry **FT_BROW_LOWERER_RIGHT** = `20`

Outer right eyebrow pulls down.

BlendShapeEntry **FT_BROW_LOWERER_LEFT** = `21`

Outer left eyebrow pulls down.

BlendShapeEntry **FT_BROW_INNER_UP_RIGHT** = `22`

Inner right eyebrow pulls up.

BlendShapeEntry **FT_BROW_INNER_UP_LEFT** = `23`

Inner left eyebrow pulls up.

BlendShapeEntry **FT_BROW_OUTER_UP_RIGHT** = `24`

Outer right eyebrow pulls up.

BlendShapeEntry **FT_BROW_OUTER_UP_LEFT** = `25`

Outer left eyebrow pulls up.

BlendShapeEntry **FT_NOSE_SNEER_RIGHT** = `26`

Right side face sneers.

BlendShapeEntry **FT_NOSE_SNEER_LEFT** = `27`

Left side face sneers.

BlendShapeEntry **FT_NASAL_DILATION_RIGHT** = `28`

Right side nose canal dilates.

BlendShapeEntry **FT_NASAL_DILATION_LEFT** = `29`

Left side nose canal dilates.

BlendShapeEntry **FT_NASAL_CONSTRICT_RIGHT** = `30`

Right side nose canal constricts.

BlendShapeEntry **FT_NASAL_CONSTRICT_LEFT** = `31`

Left side nose canal constricts.

BlendShapeEntry **FT_CHEEK_SQUINT_RIGHT** = `32`

Raises the right side cheek.

BlendShapeEntry **FT_CHEEK_SQUINT_LEFT** = `33`

Raises the left side cheek.

BlendShapeEntry **FT_CHEEK_PUFF_RIGHT** = `34`

Puffs the right side cheek.

BlendShapeEntry **FT_CHEEK_PUFF_LEFT** = `35`

Puffs the left side cheek.

BlendShapeEntry **FT_CHEEK_SUCK_RIGHT** = `36`

Sucks in the right side cheek.

BlendShapeEntry **FT_CHEEK_SUCK_LEFT** = `37`

Sucks in the left side cheek.

BlendShapeEntry **FT_JAW_OPEN** = `38`

Opens jawbone.

BlendShapeEntry **FT_MOUTH_CLOSED** = `39`

Closes the mouth.

BlendShapeEntry **FT_JAW_RIGHT** = `40`

Pushes jawbone right.

BlendShapeEntry **FT_JAW_LEFT** = `41`

Pushes jawbone left.

BlendShapeEntry **FT_JAW_FORWARD** = `42`

Pushes jawbone forward.

BlendShapeEntry **FT_JAW_BACKWARD** = `43`

Pushes jawbone backward.

BlendShapeEntry **FT_JAW_CLENCH** = `44`

Flexes jaw muscles.

BlendShapeEntry **FT_JAW_MANDIBLE_RAISE** = `45`

Raises the jawbone.

BlendShapeEntry **FT_LIP_SUCK_UPPER_RIGHT** = `46`

Upper right lip part tucks in the mouth.

BlendShapeEntry **FT_LIP_SUCK_UPPER_LEFT** = `47`

Upper left lip part tucks in the mouth.

BlendShapeEntry **FT_LIP_SUCK_LOWER_RIGHT** = `48`

Lower right lip part tucks in the mouth.

BlendShapeEntry **FT_LIP_SUCK_LOWER_LEFT** = `49`

Lower left lip part tucks in the mouth.

BlendShapeEntry **FT_LIP_SUCK_CORNER_RIGHT** = `50`

Right lip corner folds into the mouth.

BlendShapeEntry **FT_LIP_SUCK_CORNER_LEFT** = `51`

Left lip corner folds into the mouth.

BlendShapeEntry **FT_LIP_FUNNEL_UPPER_RIGHT** = 52

Upper right lip part pushes into a funnel.

BlendShapeEntry **FT_LIP_FUNNEL_UPPER_LEFT** = 53

Upper left lip part pushes into a funnel.

BlendShapeEntry **FT_LIP_FUNNEL_LOWER_RIGHT** = 54

Lower right lip part pushes into a funnel.

BlendShapeEntry **FT_LIP_FUNNEL_LOWER_LEFT** = 55

Lower left lip part pushes into a funnel.

BlendShapeEntry **FT_LIP_PUCKER_UPPER_RIGHT** = 56

Upper right lip part pushes outwards.

BlendShapeEntry **FT_LIP_PUCKER_UPPER_LEFT** = 57

Upper left lip part pushes outwards.

BlendShapeEntry **FT_LIP_PUCKER_LOWER_RIGHT** = 58

Lower right lip part pushes outwards.

BlendShapeEntry **FT_LIP_PUCKER_LOWER_LEFT** = 59

Lower left lip part pushes outwards.

BlendShapeEntry **FT_MOUTH_UPPER_UP_RIGHT** = 60

Upper right part of the lip pulls up.

BlendShapeEntry **FT_MOUTH_UPPER_UP_LEFT** = 61

Upper left part of the lip pulls up.

BlendShapeEntry **FT_MOUTH_LOWER_DOWN_RIGHT** = `62`

Lower right part of the lip pulls up.

BlendShapeEntry **FT_MOUTH_LOWER_DOWN_LEFT** = `63`

Lower left part of the lip pulls up.

BlendShapeEntry **FT_MOUTH_UPPER_DEEPEN_RIGHT** = `64`

Upper right lip part pushes in the cheek.

BlendShapeEntry **FT_MOUTH_UPPER_DEEPEN_LEFT** = `65`

Upper left lip part pushes in the cheek.

BlendShapeEntry **FT_MOUTH_UPPER_RIGHT** = `66`

Moves upper lip right.

BlendShapeEntry **FT_MOUTH_UPPER_LEFT** = `67`

Moves upper lip left.

BlendShapeEntry **FT_MOUTH_LOWER_RIGHT** = `68`

Moves lower lip right.

BlendShapeEntry **FT_MOUTH_LOWER_LEFT** = `69`

Moves lower lip left.

BlendShapeEntry **FT_MOUTH_CORNER_PULL_RIGHT** = `70`

Right lip corner pulls diagonally up and out.

BlendShapeEntry **FT_MOUTH_CORNER_PULL_LEFT** = `71`

Left lip corner pulls diagonally up and out.

BlendShapeEntry **FT_MOUTH_CORNER_SLANT_RIGHT** = `72`

Right corner lip slants up.

BlendShapeEntry **FT_MOUTH_CORNER_SLANT_LEFT** = `73`

Left corner lip slants up.

BlendShapeEntry **FT_MOUTH_FROWN_RIGHT** = `74`

Right corner lip pulls down.

BlendShapeEntry **FT_MOUTH_FROWN_LEFT** = `75`

Left corner lip pulls down.

BlendShapeEntry **FT_MOUTH_STRETCH_RIGHT** = `76`

Mouth corner lip pulls out and down.

BlendShapeEntry **FT_MOUTH_STRETCH_LEFT** = `77`

Mouth corner lip pulls out and down.

BlendShapeEntry **FT_MOUTH_DIMPLE_RIGHT** = `78`

Right lip corner is pushed backwards.

BlendShapeEntry **FT_MOUTH_DIMPLE_LEFT** = `79`

Left lip corner is pushed backwards.

BlendShapeEntry **FT_MOUTH_RAISER_UPPER** = `80`

Raises and slightly pushes out the upper mouth.

BlendShapeEntry **FT_MOUTH_RAISER_LOWER** = `81`

Raises and slightly pushes out the lower mouth.

BlendShapeEntry **FT_MOUTH_PRESS_RIGHT** = `82`

Right side lips press and flatten together vertically.

BlendShapeEntry **FT_MOUTH_PRESS_LEFT** = `83`

Left side lips press and flatten together vertically.

BlendShapeEntry **FT_MOUTH_TIGHTENER_RIGHT** = `84`

Right side lips squeeze together horizontally.

BlendShapeEntry **FT_MOUTH_TIGHTENER_LEFT** = `85`

Left side lips squeeze together horizontally.

BlendShapeEntry **FT_TONGUE_OUT** = `86`

Tongue visibly sticks out of the mouth.

BlendShapeEntry **FT_TONGUE_UP** = `87`

Tongue points upwards.

BlendShapeEntry **FT_TONGUE_DOWN** = `88`

Tongue points downwards.

BlendShapeEntry **FT_TONGUE_RIGHT** = `89`

Tongue points right.

BlendShapeEntry **FT_TONGUE_LEFT** = `90`

Tongue points left.

BlendShapeEntry **FT_TONGUE_ROLL** = `91`

Sides of the tongue funnel, creating a roll.

BlendShapeEntry **FT_TONGUE_BLEND_DOWN** = `92`

Tongue arches up then down inside the mouth.

BlendShapeEntry **FT_TONGUE_CURL_UP** = `93`

Tongue arches down then up inside the mouth.

BlendShapeEntry **FT_TONGUE_SQUISH** = `94`

Tongue squishes together and thickens.

BlendShapeEntry **FT_TONGUE_FLAT** = `95`

Tongue flattens and thins out.

BlendShapeEntry **FT_TONGUE_TWIST_RIGHT** = `96`

Tongue tip rotates clockwise, with the rest following gradually.

BlendShapeEntry **FT_TONGUE_TWIST_LEFT** = `97`

Tongue tip rotates counter-clockwise, with the rest following gradually.

BlendShapeEntry **FT_SOFT_PALATE_CLOSE** = `98`

Inner mouth throat closes.

BlendShapeEntry **FT_THROAT_SWALLOW** = `99`

The Adam's apple visibly swallows.

BlendShapeEntry **FT_NECK_FLEX_RIGHT** = `100`

Right side neck visibly flexes.

BlendShapeEntry **FT_NECK_FLEX_LEFT** = `101`

Left side neck visibly flexes.

BlendShapeEntry **FT_EYE_CLOSED** = `102`

Closes both eye lids.

BlendShapeEntry **FT_EYE_WIDE** = `103`

Widens both eye lids.

BlendShapeEntry **FT_EYE_SQUINT** = `104`

Squints both eye lids.

BlendShapeEntry **FT_EYE_DILATION** = `105`

Dilates both pupils.

BlendShapeEntry **FT_EYE_CONSTRICT** = `106`

Constricts both pupils.

BlendShapeEntry **FT_BROW_DOWN_RIGHT** = `107`

Pulls the right eyebrow down and in.

BlendShapeEntry **FT_BROW_DOWN_LEFT** = `108`

Pulls the left eyebrow down and in.

BlendShapeEntry **FT_BROW_DOWN** = `109`

Pulls both eyebrows down and in.

BlendShapeEntry **FT_BROW_UP_RIGHT** = `110`

Right brow appears worried.

BlendShapeEntry **FT_BROW_UP_LEFT** = `111`

Left brow appears worried.

BlendShapeEntry **FT_BROW_UP** = `112`

Both brows appear worried.

BlendShapeEntry **FT_NOSE_SNEER** = `113`

Entire face sneers.

BlendShapeEntry **FT_NASAL_DILATION** = `114`

Both nose canals dilate.

BlendShapeEntry **FT_NASAL_CONSTRICT** = `115`

Both nose canals constrict.

BlendShapeEntry **FT_CHEEK_PUFF** = `116`

Puffs both cheeks.

BlendShapeEntry **FT_CHEEK_SUCK** = `117`

Sucks in both cheeks.

BlendShapeEntry **FT_CHEEK_SQUINT** = `118`

Raises both cheeks.

BlendShapeEntry **FT_LIP_SUCK_UPPER** = `119`

Tucks in the upper lips.

BlendShapeEntry **FT_LIP_SUCK_LOWER** = `120`

Tucks in the lower lips.

BlendShapeEntry **FT_LIP_SUCK** = `121`

Tucks in both lips.

BlendShapeEntry **FT_LIP_FUNNEL_UPPER** = `122`

Funnels in the upper lips.

BlendShapeEntry **FT_LIP_FUNNEL_LOWER** = `123`

Funnels in the lower lips.

BlendShapeEntry **FT_LIP_FUNNEL** = `124`

Funnels in both lips.

BlendShapeEntry **FT_LIP_PUCKER_UPPER** = `125`

Upper lip part pushes outwards.

BlendShapeEntry **FT_LIP_PUCKER_LOWER** = `126`

Lower lip part pushes outwards.

BlendShapeEntry **FT_LIP_PUCKER** = `127`

Lips push outwards.

BlendShapeEntry **FT_MOUTH_UPPER_UP** = `128`

Raises the upper lips.

BlendShapeEntry **FT_MOUTH_LOWER_DOWN** = `129`

Lowers the lower lips.

BlendShapeEntry **FT_MOUTH_OPEN** = `130`

Mouth opens, revealing teeth.

BlendShapeEntry **FT_MOUTH_RIGHT** = `131`

Moves mouth right.

BlendShapeEntry **FT_MOUTH_LEFT** = `132`

Moves mouth left.

BlendShapeEntry **FT_MOUTH_SMILE_RIGHT** = `133`

Right side of the mouth smiles.

BlendShapeEntry **FT_MOUTH_SMILE_LEFT** = `134`

Left side of the mouth smiles.

BlendShapeEntry **FT_MOUTH_SMILE** = `135`

Mouth expresses a smile.

BlendShapeEntry **FT_MOUTH_SAD_RIGHT** = `136`

Right side of the mouth expresses sadness.

BlendShapeEntry **FT_MOUTH_SAD_LEFT** = `137`

Left side of the mouth expresses sadness.

BlendShapeEntry **FT_MOUTH_SAD** = `138`

Mouth expresses sadness.

BlendShapeEntry **FT_MOUTH_STRETCH** = `139`

Mouth stretches.

BlendShapeEntry **FT_MOUTH_DIMPLE** = `140`

Lip corners dimple.

BlendShapeEntry **FT_MOUTH_TIGHTENER** = `141`

Mouth tightens.

BlendShapeEntry **FT_MOUTH_PRESS** = `142`

Mouth presses together.

BlendShapeEntry **FT_MAX** = `143`

Represents the size of the BlendShapeEntry enum.

Property Descriptions

PackedFloat32Array **blend_shapes** = `PackedFloat32Array()` 🔗

- void **set_blend_shapes**(value: PackedFloat32Array)
- PackedFloat32Array **get_blend_shapes**()

The array of face blend shape weights with indices corresponding to the BlendShapeEntry enum.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedFloat32Array for more details.

Method Descriptions

float **get_blend_shape**(blend_shape: BlendShapeEntry) const

Returns the requested face blend shape weight.

void **set_blend_shape**(blend_shape: BlendShapeEntry, weight: float)

Sets a face blend shape weight.

XRHandTracker

Inherits: XRPositionalTracker **<** XRTracker **<** RefCounted **<** Object

A tracked hand in XR.

Description

A hand tracking system will create an instance of this object and add it to the XRServer. This tracking system will then obtain skeleton data, convert it to the Godot Humanoid hand skeleton and store this data on the **XRHandTracker** object.

Use XRHandModifier3D to animate a hand mesh using hand tracking data.

Tutorials

- XR documentation index

Properties

TrackerHand	hand	`1` (overrides XRPositionalTracker)
HandTrackingSource	hand_tracking_source	`0`
bool	has_tracking_data	`false`

TrackerType	type	`16` (overrides XRTracker)

Methods

Vector3	get_hand_joint_angular_velocity(joint: HandJoint) const
BitField[HandJointFlags]	get_hand_joint_flags(joint: HandJoint) const
Vector3	get_hand_joint_linear_velocity(joint: HandJoint) const
float	get_hand_joint_radius(joint: HandJoint) const
Transform3D	get_hand_joint_transform(joint: HandJoint) const
void	set_hand_joint_angular_velocity(joint: HandJoint, angular_velocity: Vector3)
void	set_hand_joint_flags(joint: HandJoint, flags: BitField[HandJointFlags])

void	set_hand_joint_linear_velocity(joint: HandJoint, linear_velocity: Vector3)
void	set_hand_joint_radius(joint: HandJoint, radius: float)
void	set_hand_joint_transform(joint: HandJoint, transform: Transform3D)

Enumerations

enum **HandTrackingSource**: 🔗

HandTrackingSource **HAND_TRACKING_SOURCE_UNKNOWN** = `0`

The source of hand tracking data is unknown.

HandTrackingSource **HAND_TRACKING_SOURCE_UNOBSTRUCTED** = `1`

The source of hand tracking data is unobstructed, meaning that an accurate method of hand tracking is used. These include optical hand tracking, data gloves, etc.

HandTrackingSource **HAND_TRACKING_SOURCE_CONTROLLER** = `2`

The source of hand tracking data is a controller, meaning that joint positions are inferred from controller inputs.

HandTrackingSource **HAND_TRACKING_SOURCE_MAX** = `3`

Represents the size of the HandTrackingSource enum.

---

enum **HandJoint**: 🔗

HandJoint **HAND_JOINT_PALM** = `0`

Palm joint.

HandJoint **HAND_JOINT_WRIST** = `1`

Wrist joint.

HandJoint **HAND_JOINT_THUMB_METACARPAL** = `2`

Thumb metacarpal joint.

HandJoint **HAND_JOINT_THUMB_PHALANX_PROXIMAL** = `3`

Thumb phalanx proximal joint.

HandJoint **HAND_JOINT_THUMB_PHALANX_DISTAL** = `4`

Thumb phalanx distal joint.

HandJoint **HAND_JOINT_THUMB_TIP** = `5`

Thumb tip joint.

HandJoint **HAND_JOINT_INDEX_FINGER_METACARPAL** = `6`

Index finger metacarpal joint.

HandJoint **HAND_JOINT_INDEX_FINGER_PHALANX_PROXIMAL** = `7`

Index finger phalanx proximal joint.

HandJoint **HAND_JOINT_INDEX_FINGER_PHALANX_INTERMEDIATE** = `8`

Index finger phalanx intermediate joint.

HandJoint **HAND_JOINT_INDEX_FINGER_PHALANX_DISTAL** = `9`

Index finger phalanx distal joint.

HandJoint **HAND_JOINT_INDEX_FINGER_TIP** = `10`

Index finger tip joint.

HandJoint **HAND_JOINT_MIDDLE_FINGER_METACARPAL** = `11`

Middle finger metacarpal joint.

HandJoint **HAND_JOINT_MIDDLE_FINGER_PHALANX_PROXIMAL** = `12`

Middle finger phalanx proximal joint.

HandJoint **HAND_JOINT_MIDDLE_FINGER_PHALANX_INTERMEDIATE** = `13`

Middle finger phalanx intermediate joint.

HandJoint **HAND_JOINT_MIDDLE_FINGER_PHALANX_DISTAL** = `14`

Middle finger phalanx distal joint.

HandJoint **HAND_JOINT_MIDDLE_FINGER_TIP** = `15`

Middle finger tip joint.

HandJoint **HAND_JOINT_RING_FINGER_METACARPAL** = `16`

Ring finger metacarpal joint.

HandJoint **HAND_JOINT_RING_FINGER_PHALANX_PROXIMAL** = `17`

Ring finger phalanx proximal joint.

HandJoint **HAND_JOINT_RING_FINGER_PHALANX_INTERMEDIATE** = `18`

Ring finger phalanx intermediate joint.

HandJoint **HAND_JOINT_RING_FINGER_PHALANX_DISTAL** = `19`

Ring finger phalanx distal joint.

HandJoint **HAND_JOINT_RING_FINGER_TIP** = `20`

Ring finger tip joint.

HandJoint **HAND_JOINT_PINKY_FINGER_METACARPAL** = `21`

Pinky finger metacarpal joint.

HandJoint **HAND_JOINT_PINKY_FINGER_PHALANX_PROXIMAL** = `22`

Pinky finger phalanx proximal joint.

HandJoint **HAND_JOINT_PINKY_FINGER_PHALANX_INTERMEDIATE** = `23`

Pinky finger phalanx intermediate joint.

HandJoint **HAND_JOINT_PINKY_FINGER_PHALANX_DISTAL** = `24`

Pinky finger phalanx distal joint.

HandJoint **HAND_JOINT_PINKY_FINGER_TIP** = `25`

Pinky finger tip joint.

HandJoint **HAND_JOINT_MAX** = `26`

Represents the size of the HandJoint enum.

flags **HandJointFlags**: 🔗

HandJointFlags **HAND_JOINT_FLAG_ORIENTATION_VALID** = `1`

The hand joint's orientation data is valid.

HandJointFlags **HAND_JOINT_FLAG_ORIENTATION_TRACKED** = `2`

The hand joint's orientation is actively tracked. May not be set if tracking has been temporarily lost.

HandJointFlags **HAND_JOINT_FLAG_POSITION_VALID** = `4`

The hand joint's position data is valid.

HandJointFlags **HAND_JOINT_FLAG_POSITION_TRACKED** = `8`

The hand joint's position is actively tracked. May not be set if tracking has been temporarily lost.

HandJointFlags **HAND_JOINT_FLAG_LINEAR_VELOCITY_VALID** = `16`

The hand joint's linear velocity data is valid.

HandJointFlags **HAND_JOINT_FLAG_ANGULAR_VELOCITY_VALID** = `32`

The hand joint's angular velocity data is valid.

Property Descriptions

HandTrackingSource **hand_tracking_source** = `0`

- void **set_hand_tracking_source**(value: HandTrackingSource)
- HandTrackingSource **get_hand_tracking_source**()

The source of the hand tracking data.

---

bool **has_tracking_data** = `false`

- void **set_has_tracking_data**(value: bool)
- bool **get_has_tracking_data**()

If `true`, the hand tracking data is valid.

Method Descriptions

Vector3 **get_hand_joint_angular_velocity**(joint: HandJoint) const

Returns the angular velocity for the given hand joint.

BitField[HandJointFlags] **get_hand_joint_flags**(joint: HandJoint) const

Returns flags about the validity of the tracking data for the given hand joint (see HandJointFlags).

Vector3 **get_hand_joint_linear_velocity**(joint: HandJoint) const

Returns the linear velocity for the given hand joint.

---

float **get_hand_joint_radius**(joint: HandJoint) const

Returns the radius of the given hand joint.

Transform3D **get_hand_joint_transform**(joint: HandJoint) const

Returns the transform for the given hand joint.

---

void **set_hand_joint_angular_velocity**(joint: HandJoint, angular_velocity: Vector3)

Sets the angular velocity for the given hand joint.

---

void **set_hand_joint_flags**(joint: HandJoint, flags: BitField[HandJointFlags])

Sets flags about the validity of the tracking data for the given hand joint.

---

void **set_hand_joint_linear_velocity**(joint: HandJoint, linear_velocity: Vector3) 🔗

Sets the linear velocity for the given hand joint.

---

void **set_hand_joint_radius**(joint: HandJoint, radius: float) 🔗

Sets the radius of the given hand joint.

---

void **set_hand_joint_transform**(joint: HandJoint, transform: Transform3D) 🔗

Sets the transform for the given hand joint.

XRInterface

Inherits: RefCounted **<** Object

Inherited By: MobileVRInterface, OpenXRInterface, WebXRInterface, XRInterfaceExtension

Base class for an XR interface implementation.

Description

This class needs to be implemented to make an AR or VR platform available to Godot and these should be implemented as C++ modules or GDExtension modules. Part of the interface is exposed to GDScript so you can detect, enable and configure an AR or VR platform.

Interfaces should be written in such a way that simply enabling them will give us a working setup. You can query the available interfaces through XRServer.

Tutorials

- XR documentation index

Properties

bool	ar_is_anchor_detection_enabled	`false`

EnvironmentBlendMode	environment_blend_mode	`0`
bool	interface_is_primary	`false`
PlayAreaMode	xr_play_area_mode	`0`

Methods

int	get_camera_feed_id()
int	get_capabilities() const
StringName	get_name() const
PackedVector3Array	get_play_area() const
Projection	get_projection_for_view(view: int, aspect: float, near: float, far: float)
Vector2	get_render_target_size()
Array	get_supported_environment_blend_modes()
Dictionary	get_system_info()

TrackingStatus	get_tracking_status() const
Transform3D	get_transform_for_view(view: int, cam_transform: Transform3D)
int	get_view_count()
bool	initialize()
bool	is_initialized() const
bool	is_passthrough_enabled()
bool	is_passthrough_supported()
bool	set_environment_blend_mode(mode: EnvironmentBlendMode)
bool	set_play_area_mode(mode: PlayAreaMode)
bool	start_passthrough()
void	stop_passthrough()
bool	supports_play_area_mode(mode: PlayAreaMode)

void	trigger_haptic_pulse(action_name: String, tracker_name: StringName, frequency: float, amplitude: float, duration_sec: float, delay_sec: float)
void	uninitialize()

Signals

play_area_changed(mode: int) 🔗

Emitted when the play area is changed. This can be a result of the player resetting the boundary or entering a new play area, the player changing the play area mode, the world scale changing or the player resetting their headset orientation.

Enumerations

enum **Capabilities**: 🔗

Capabilities **XR_NONE** = `0`

No XR capabilities.

Capabilities **XR_MONO** = `1`

This interface can work with normal rendering output (non-HMD based AR).

Capabilities **XR_STEREO** = `2`

This interface supports stereoscopic rendering.

Capabilities **XR_QUAD** = `4`

This interface supports quad rendering (not yet supported by Godot).

Capabilities **XR_VR** = `8`

This interface supports VR.

Capabilities **XR_AR** = `16`

This interface supports AR (video background and real world tracking).

Capabilities **XR_EXTERNAL** = `32`

This interface outputs to an external device. If the main viewport is used, the on screen output is an unmodified buffer of either the left or right eye (stretched if the viewport size is not changed to the same aspect ratio of get_render_target_size). Using a separate viewport node frees up the main viewport for other purposes.

enum **TrackingStatus**: 🔗

TrackingStatus **XR_NORMAL_TRACKING** = `0`

Tracking is behaving as expected.

TrackingStatus **XR_EXCESSIVE_MOTION** = `1`

Tracking is hindered by excessive motion (the player is moving faster than tracking can keep up).

TrackingStatus **XR_INSUFFICIENT_FEATURES** = `2`

Tracking is hindered by insufficient features, it's too dark (for camera-based tracking), player is blocked, etc.

TrackingStatus **XR_UNKNOWN_TRACKING** = `3`

We don't know the status of the tracking or this interface does not provide feedback.

TrackingStatus **XR_NOT_TRACKING** = `4`

Tracking is not functional (camera not plugged in or obscured, lighthouses turned off, etc.).

---

enum **PlayAreaMode**: 🔗

PlayAreaMode **XR_PLAY_AREA_UNKNOWN** = `0`

Play area mode not set or not available.

PlayAreaMode **XR_PLAY_AREA_3DOF** = `1`

Play area only supports orientation tracking, no positional tracking, area will center around player.

PlayAreaMode **XR_PLAY_AREA_SITTING** = `2`

Player is in seated position, limited positional tracking, fixed guardian around player.

PlayAreaMode **XR_PLAY_AREA_ROOMSCALE** = `3`

Player is free to move around, full positional tracking.

PlayAreaMode **XR_PLAY_AREA_STAGE** = `4`

Same as XR_PLAY_AREA_ROOMSCALE but origin point is fixed to the center of the physical space. In this mode, system-level recentering may be disabled, requiring the use of XRServer.center_on_hmd.

enum **EnvironmentBlendMode**: 🔗

EnvironmentBlendMode **XR_ENV_BLEND_MODE_OPAQUE** = `0`

Opaque blend mode. This is typically used for VR devices.

EnvironmentBlendMode **XR_ENV_BLEND_MODE_ADDITIVE** = `1`

Additive blend mode. This is typically used for AR devices or VR devices with passthrough.

EnvironmentBlendMode **XR_ENV_BLEND_MODE_ALPHA_BLEND** = `2`

Alpha blend mode. This is typically used for AR or VR devices with passthrough capabilities. The alpha channel controls how much of the passthrough is visible. Alpha of 0.0 means the passthrough is visible and this pixel works in ADDITIVE mode. Alpha of 1.0 means that the passthrough is not visible and this pixel works in OPAQUE mode.

Property Descriptions

bool **ar_is_anchor_detection_enabled** = `false` 🔗

- void **set_anchor_detection_is_enabled**(value: bool)
- bool **get_anchor_detection_is_enabled**()

On an AR interface, `true` if anchor detection is enabled.

EnvironmentBlendMode **environment_blend_mode** = `0` 🔗

- bool **set_environment_blend_mode**(mode: EnvironmentBlendMode)
- EnvironmentBlendMode **get_environment_blend_mode**()

Specify how XR should blend in the environment. This is specific to certain AR and passthrough devices where camera images are blended in by the XR compositor.

---

bool **interface_is_primary** = `false` 🔗

- void **set_primary**(value: bool)
- bool **is_primary**()

`true` if this is the primary interface.

PlayAreaMode **xr_play_area_mode** = `0` 🔗

- bool **set_play_area_mode**(mode: PlayAreaMode)
- PlayAreaMode **get_play_area_mode**()

The play area mode for this interface.

Method Descriptions

int **get_camera_feed_id**() 🔗

If this is an AR interface that requires displaying a camera feed as the background, this method returns the feed ID in the CameraServer for this interface.

---

int **get_capabilities**() const

Returns a combination of Capabilities flags providing information about the capabilities of this interface.

StringName **get_name**() const

Returns the name of this interface (`"OpenXR"`, `"OpenVR"`, `"OpenHMD"`, `"ARKit"`, etc.).

PackedVector3Array **get_play_area**() const

Returns an array of vectors that represent the physical play area mapped to the virtual space around the XROrigin3D point. The points form a convex polygon that can be used to react to or visualize the play area. This returns an empty array if this feature is not supported or if the information is not yet available.

Projection **get_projection_for_view**(view: int, aspect: float, near: float, far: float)

Returns the projection matrix for a view/eye.

Vector2 **get_render_target_size**()

Returns the resolution at which we should render our intermediate results before things like lens distortion are applied by the VR platform.

Array **get_supported_environment_blend_modes**()

Returns the an array of supported environment blend modes, see EnvironmentBlendMode.

Dictionary **get_system_info**()

Returns a Dictionary with extra system info. Interfaces are expected to return `XRRuntimeName` and `XRRuntimeVersion` providing info about the used XR runtime. Additional entries may be provided specific to an interface.

Note: This information may only be available after initialize was successfully called.

TrackingStatus **get_tracking_status**() const

If supported, returns the status of our tracking. This will allow you to provide feedback to the user whether there are issues with positional tracking.

Transform3D **get_transform_for_view**(view: int, cam_transform: Transform3D)

Returns the transform for a view/eye.

`view` is the view/eye index.

`cam_transform` is the transform that maps device coordinates to scene coordinates, typically the Node3D.global_transform of the current XROrigin3D.

---

int **get_view_count**()

Returns the number of views that need to be rendered for this device. 1 for Monoscopic, 2 for Stereoscopic.

---

bool **initialize**() 🔗

Call this to initialize this interface. The first interface that is initialized is identified as the primary interface and it will be used for rendering output.

After initializing the interface you want to use you then need to enable the AR/VR mode of a viewport and rendering should commence.

Note: You must enable the XR mode on the main viewport for any device that uses the main output of Godot, such as for mobile VR.

If you do this for a platform that handles its own output (such as OpenVR) Godot will show just one eye without distortion on screen. Alternatively, you can add a separate viewport node to your scene and enable AR/VR on that viewport. It will be used to output to the HMD, leaving you free to do anything you like in the main window, such as using a separate camera as a spectator camera or rendering something completely different.

While currently not used, you can activate additional interfaces. You may wish to do this if you want to track controllers from other platforms. However, at this point in time only one interface can render to an HMD.

---

bool **is_initialized**() const 🔗

Returns `true` if this interface has been initialized.

---

bool **is_passthrough_enabled**() 🔗

Deprecated: Check if environment_blend_mode is XR_ENV_BLEND_MODE_ALPHA_BLEND, instead.

Returns `true` if passthrough is enabled.

---

bool **is_passthrough_supported**()

Deprecated: Check that XR_ENV_BLEND_MODE_ALPHA_BLEND is supported using get_supported_environment_blend_modes, instead.

Returns `true` if this interface supports passthrough.

---

bool **set_environment_blend_mode**(mode: EnvironmentBlendMode)

Sets the active environment blend mode.

`mode` is the environment blend mode starting with the next frame.

Note: Not all runtimes support all environment blend modes, so it is important to check this at startup. For example:

```
func _ready():
    var xr_interface: XRInterface =
XRServer.find_interface("OpenXR")
    if xr_interface and xr_interface.is_initialized():
        var vp: Viewport = get_viewport()
        vp.use_xr = true
        var acceptable_modes =
[XRInterface.XR_ENV_BLEND_MODE_OPAQUE,
XRInterface.XR_ENV_BLEND_MODE_ADDITIVE]
        var modes =
xr_interface.get_supported_environment_blend_modes()
        for mode in acceptable_modes:
            if mode in modes:
                xr_interface.set_environment_blend_mode(mode)
                break
```

---

bool **set_play_area_mode**(mode: PlayAreaMode)

Sets the active play area mode, will return `false` if the mode can't be used with this interface.

Note: Changing this after the interface has already been initialized can be jarring for the player, so it's recommended to recenter on the HMD with XRServer.center_on_hmd (if switching to XR_PLAY_AREA_STAGE) or make the switch during a scene change.

---

bool **start_passthrough**()

Deprecated: Set the environment_blend_mode to XR_ENV_BLEND_MODE_ALPHA_BLEND, instead.

Starts passthrough, will return `false` if passthrough couldn't be started.

Note: The viewport used for XR must have a transparent background, otherwise passthrough may not properly render.

---

void **stop_passthrough**()

Deprecated: Set the environment_blend_mode to XR_ENV_BLEND_MODE_OPAQUE, instead.

Stops passthrough.

---

bool **supports_play_area_mode**(mode: PlayAreaMode)

Call this to find out if a given play area mode is supported by this interface.

---

void **trigger_haptic_pulse**(action_name: String, tracker_name: StringName, frequency: float, amplitude: float, duration_sec: float, delay_sec: float) 🔗

Triggers a haptic pulse on a device associated with this interface.

`action_name` is the name of the action for this pulse.

`tracker_name` is optional and can be used to direct the pulse to a specific device provided that device is bound to this haptic.

`frequency` is the frequency of the pulse, set to `0.0` to have the system use a default frequency.

`amplitude` is the amplitude of the pulse between `0.0` and `1.0`.

`duration_sec` is the duration of the pulse in seconds.

`delay_sec` is a delay in seconds before the pulse is given.

---

void **uninitialize**() 🔗

Turns the interface off.

XRInterfaceExtension

Inherits: XRInterface **<** RefCounted **<** Object

Base class for XR interface extensions (plugins).

Description

External XR interface plugins should inherit from this class.

Tutorials

- XR documentation index

Methods

void	_end_frame() virtual
bool	_get_anchor_detection_is_enabled() virtual const
int	_get_camera_feed_id() virtual const
Transform3D	_get_camera_transform() virtual
int	_get_capabilities() virtual const

RID	_get_color_texture() virtual
RID	_get_depth_texture() virtual
StringName	_get_name() virtual const
PackedVector3Array	_get_play_area() virtual const
PlayAreaMode	_get_play_area_mode() virtual const
PackedFloat64Array	_get_projection_for_view(view: int, aspect: float, z_near: float, z_far: float) virtual
Vector2	_get_render_target_size() virtual
PackedStringArray	_get_suggested_pose_names(tracker_name: StringName) virtual const
PackedStringArray	_get_suggested_tracker_names() virtual const
Dictionary	_get_system_info() virtual const
TrackingStatus	_get_tracking_status() virtual const
Transform3D	_get_transform_for_view(view: int, cam_transform: Transform3D) virtual

RID	_get_velocity_texture() virtual
int	_get_view_count() virtual
RID	_get_vrs_texture() virtual
bool	_initialize() virtual
bool	_is_initialized() virtual const
void	_post_draw_viewport(render_target: RID, screen_rect: Rect2) virtual
bool	_pre_draw_viewport(render_target: RID) virtual
void	_pre_render() virtual
void	_process() virtual
void	_set_anchor_detection_is_enabled(enabled: bool) virtual
bool	_set_play_area_mode(mode: PlayAreaMode) virtual const

bool	_supports_play_area_mode(mode: PlayAreaMode) virtual const
void	_trigger_haptic_pulse(action_name: String, tracker_name: StringName, frequency: float, amplitude: float, duration_sec: float, delay_sec: float) virtual
void	_uninitialize() virtual
void	add_blit(render_target: RID, src_rect: Rect2, dst_rect: Rect2i, use_layer: bool, layer: int, apply_lens_distortion: bool, eye_center: Vector2, k1: float, k2: float, upscale: float, aspect_ratio: float)
RID	get_color_texture()
RID	get_depth_texture()
RID	get_render_target_texture(render_target: RID)
RID	get_velocity_texture()

Method Descriptions

void **_end_frame**() virtual 🔗

Called if interface is active and queues have been submitted.

---

bool **_get_anchor_detection_is_enabled**() virtual const 🔗

Return `true` if anchor detection is enabled for this interface.

---

int **_get_camera_feed_id**() virtual const 🔗

Returns the camera feed ID for the CameraFeed registered with the CameraServer that should be presented as the background on an AR capable device (if applicable).

Transform3D **_get_camera_transform**() virtual 🔗

Returns the Transform3D that positions the XRCamera3D in the world.

---

int **_get_capabilities**() virtual const 🔗

Returns the capabilities of this interface.

RID **_get_color_texture**() virtual 🔗

Return color texture into which to render (if applicable).

RID **_get_depth_texture**() virtual 🔗

Return depth texture into which to render (if applicable).

StringName **_get_name**() virtual const 🔗

Returns the name of this interface.

PackedVector3Array **_get_play_area**() virtual const 🔗

Returns a PackedVector3Array that represents the play areas boundaries (if applicable).

PlayAreaMode **_get_play_area_mode**() virtual const 🔗

Returns the play area mode that sets up our play area.

PackedFloat64Array **_get_projection_for_view**(view: int, aspect: float, z_near: float, z_far: float) virtual 🔗

Returns the projection matrix for the given view as a PackedFloat64Array.

Vector2 **_get_render_target_size**() virtual 🔗

Returns the size of our render target for this interface, this overrides the size of the Viewport marked as the xr viewport.

PackedStringArray **_get_suggested_pose_names**(tracker_name: StringName) virtual const 🔗

Returns a PackedStringArray with pose names configured by this interface. Note that user configuration can override this list.

PackedStringArray **_get_suggested_tracker_names**() virtual const

Returns a PackedStringArray with tracker names configured by this interface. Note that user configuration can override this list.

Dictionary **_get_system_info**() virtual const

Returns a Dictionary with system information related to this interface.

TrackingStatus **_get_tracking_status**() virtual const

Returns a TrackingStatus specifying the current status of our tracking.

Transform3D **_get_transform_for_view**(view: int, cam_transform: Transform3D) virtual

Returns a Transform3D for a given view.

RID **_get_velocity_texture**() virtual

Return velocity texture into which to render (if applicable).

int **_get_view_count**() virtual

Returns the number of views this interface requires, 1 for mono, 2 for stereoscopic.

RID **_get_vrs_texture**() virtual

There is currently no description for this method. Please help us by contributing one!

---

bool **_initialize**() virtual 🔗

Initializes the interface, returns `true` on success.

---

bool **_is_initialized**() virtual const 🔗

Returns `true` if this interface has been initialized.

---

void **_post_draw_viewport**(render_target: RID, screen_rect: Rect2) virtual 🔗

Called after the XR Viewport draw logic has completed.

---

bool **_pre_draw_viewport**(render_target: RID) virtual 🔗

Called if this is our primary **XRInterfaceExtension** before we start processing a Viewport for every active XR Viewport, returns `true` if that viewport should be rendered. An XR interface may return `false` if the user has taken off their headset and we can pause rendering.

---

void **_pre_render**() virtual 🔗

Called if this **XRInterfaceExtension** is active before rendering starts. Most XR interfaces will sync tracking at this point in time.

---

void **_process**() virtual 🔗

Called if this **XRInterfaceExtension** is active before our physics and game process is called. Most XR interfaces will update its XRPositionalTrackers at this point in time.

void **_set_anchor_detection_is_enabled**(enabled: bool) virtual 🔗

Enables anchor detection on this interface if supported.

bool **_set_play_area_mode**(mode: PlayAreaMode) virtual const 🔗

Set the play area mode for this interface.

bool **_supports_play_area_mode**(mode: PlayAreaMode) virtual const 🔗

Returns `true` if this interface supports this play area mode.

void **_trigger_haptic_pulse**(action_name: String, tracker_name: StringName, frequency: float, amplitude: float, duration_sec: float, delay_sec: float) virtual 🔗

Triggers a haptic pulse to be emitted on the specified tracker.

void **_uninitialize**() virtual 🔗

Uninitialize the interface.

void **add_blit**(render_target: RID, src_rect: Rect2, dst_rect: Rect2i, use_layer: bool, layer: int, apply_lens_distortion: bool, eye_center:

Vector2, k1: float, k2: float, upscale: float, aspect_ratio: float) 🔗

Blits our render results to screen optionally applying lens distortion. This can only be called while processing `_commit_views`.

RID **get_color_texture**() 🔗

There is currently no description for this method. Please help us by contributing one!

RID **get_depth_texture**() 🔗

There is currently no description for this method. Please help us by contributing one!

RID **get_render_target_texture**(render_target: RID) 🔗

Returns a valid RID for a texture to which we should render the current frame if supported by the interface.

RID **get_velocity_texture**() 🔗

There is currently no description for this method. Please help us by contributing one!

XRPose

Inherits: RefCounted **<** Object

This object contains all data related to a pose on a tracked object.

Description

XR runtimes often identify multiple locations on devices such as controllers that are spatially tracked.

Orientation, location, linear velocity and angular velocity are all provided for each pose by the XR runtime. This object contains this state of a pose.

Tutorials

- XR documentation index

Properties

Vector3	angular_velocity	`Vector3(0, 0, 0)`
bool	has_tracking_data	`false`
Vector3	linear_velocity	`Vector3(0, 0, 0)`

StringName	name	`&""`
TrackingConfidence	tracking_confidence	`0`
Transform3D	transform	`Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`

Methods

Transform3D	get_adjusted_transform() const

Enumerations

enum **TrackingConfidence**: 🔗

TrackingConfidence **XR_TRACKING_CONFIDENCE_NONE** = `0`

No tracking information is available for this pose.

TrackingConfidence **XR_TRACKING_CONFIDENCE_LOW** = `1`

Tracking information may be inaccurate or estimated. For example, with inside out tracking this would indicate a controller may be (partially) obscured.

TrackingConfidence **XR_TRACKING_CONFIDENCE_HIGH** = `2`

Tracking information is considered accurate and up to date.

Property Descriptions

Vector3 **angular_velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_angular_velocity**(value: Vector3)
- Vector3 **get_angular_velocity**()

The angular velocity for this pose.

---

bool **has_tracking_data** = `false` 🔗

- void **set_has_tracking_data**(value: bool)
- bool **get_has_tracking_data**()

If `true` our tracking data is up to date. If `false` we're no longer receiving new tracking data and our state is whatever that last valid state was.

Vector3 **linear_velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_linear_velocity**(value: Vector3)
- Vector3 **get_linear_velocity**()

The linear velocity of this pose.

StringName **name** = `&""` 🔗

- void **set_name**(value: StringName)
- StringName **get_name**()

The name of this pose. Pose names are often driven by an action map setup by the user. Godot does suggest a number of pose names that it expects XRInterfaces to implement:

- `root` defines a root location, often used for tracked objects that do not have further nodes.
- `aim` defines the tip of a controller with the orientation pointing outwards, for example: add your raycasts to this.
- `grip` defines the location where the user grips the controller
- `skeleton` defines the root location a hand mesh should be placed when using hand tracking and the animated skeleton supplied by the XR runtime.

TrackingConfidence **tracking_confidence** = `0` 🔗

- void **set_tracking_confidence**(value: TrackingConfidence)
- TrackingConfidence **get_tracking_confidence**()

The tracking confidence for this pose, provides insight on how accurate the spatial positioning of this record is.

Transform3D **transform** = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)` 🔗

- void **set_transform**(value: Transform3D)
- Transform3D **get_transform**()

The transform containing the original and transform as reported by the XR runtime.

Method Descriptions

Transform3D **get_adjusted_transform**() const 🔗

Returns the transform with world scale and our reference frame applied. This is the transform used to position XRNode3D objects.

XRPositionalTracker

Inherits: XRTracker **<** RefCounted **<** Object

Inherited By: XRBodyTracker, XRControllerTracker, XRHandTracker

A tracked object.

Description

An instance of this object represents a device that is tracked, such as a controller or anchor point. HMDs aren't represented here as they are handled internally.

As controllers are turned on and the XRInterface detects them, instances of this object are automatically added to this list of active tracking objects accessible through the XRServer.

The XRNode3D and XRAnchor3D both consume objects of this type and should be used in your project. The positional trackers are just under-the-hood objects that make this all work. These are mostly exposed so that GDExtension-based interfaces can interact with them.

Tutorials

- XR documentation index

Properties

TrackerHand	hand	`0`
String	profile	`""`

Methods

Variant	get_input(name: StringName) const
XRPose	get_pose(name: StringName) const
bool	has_pose(name: StringName) const
void	invalidate_pose(name: StringName)
void	set_input(name: StringName, value: Variant)
void	set_pose(name: StringName, transform: Transform3D, linear_velocity: Vector3, angular_velocity: Vector3, tracking_confidence: TrackingConfidence)

Signals

button_pressed(name: String) 🔗

Emitted when a button on this tracker is pressed. Note that many XR runtimes allow other inputs to be mapped to buttons.

---

button_released(name: String)

Emitted when a button on this tracker is released.

---

input_float_changed(name: String, value: float)

Emitted when a trigger or similar input on this tracker changes value.

---

input_vector2_changed(name: String, vector: Vector2)

Emitted when a thumbstick or thumbpad on this tracker moves.

---

pose_changed(pose: XRPose)

Emitted when the state of a pose tracked by this tracker changes.

---

pose_lost_tracking(pose: XRPose)

Emitted when a pose tracked by this tracker stops getting updated tracking data.

---

profile_changed(role: String)

Emitted when the profile of our tracker changes.

Enumerations

enum **TrackerHand**:

TrackerHand **TRACKER_HAND_UNKNOWN** = `0`

The hand this tracker is held in is unknown or not applicable.

TrackerHand **TRACKER_HAND_LEFT** = `1`

This tracker is the left hand controller.

TrackerHand **TRACKER_HAND_RIGHT** = `2`

This tracker is the right hand controller.

TrackerHand **TRACKER_HAND_MAX** = `3`

Represents the size of the TrackerHand enum.

Property Descriptions

TrackerHand **hand** = `0`

- void **set_tracker_hand**(value: TrackerHand)
- TrackerHand **get_tracker_hand**()

Defines which hand this tracker relates to.

String **profile** = `""`

- void **set_tracker_profile**(value: String)
- String **get_tracker_profile**()

The profile associated with this tracker, interface dependent but will indicate the type of controller being tracked.

Method Descriptions

Variant **get_input**(name: StringName) const 🔗

Deprecated: Use through XRControllerTracker.

Returns an input for this tracker. It can return a boolean, float or Vector2 value depending on whether the input is a button, trigger or thumbstick/thumbpad.

XRPose **get_pose**(name: StringName) const 🔗

Returns the current XRPose state object for the bound `name` pose.

bool **has_pose**(name: StringName) const 🔗

Returns `true` if the tracker is available and is currently tracking the bound `name` pose.

void **invalidate_pose**(name: StringName) 🔗

Marks this pose as invalid, we don't clear the last reported state but it allows users to decide if trackers need to be hidden if we lose tracking or just remain at their last known position.

void **set_input**(name: StringName, value: Variant) 🔗

Deprecated: Use through XRControllerTracker.

Changes the value for the given input. This method is called by a XRInterface implementation and should not be used directly.

---

void **set_pose**(name: StringName, transform: Transform3D, linear_velocity: Vector3, angular_velocity: Vector3, tracking_confidence: TrackingConfidence) 🔗

Sets the transform, linear velocity, angular velocity and tracking confidence for the given pose. This method is called by a XRInterface implementation and should not be used directly.

XRServer

Inherits: Object

Server for AR and VR features.

Description

The AR/VR server is the heart of our Advanced and Virtual Reality solution and handles all the processing.

Tutorials

- XR documentation index

Properties

XRInterface	primary_interface	
Transform3D	world_origin	`Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`
float	world_scale	`1.0`

Methods

void	add_interface(interface: XRInterface)
void	add_tracker(tracker: XRTracker)
void	center_on_hmd(rotation_mode: RotationMode, keep_height: bool)
void	clear_reference_frame()
XRInterface	find_interface(name: String) const
Transform3D	get_hmd_transform()
XRInterface	get_interface(idx: int) const
int	get_interface_count() const
Array[Dictionary]	get_interfaces() const
Transform3D	get_reference_frame() const
XRTracker	get_tracker(tracker_name: StringName) const
Dictionary	get_trackers(tracker_types: int)

void	remove_interface(interface: XRInterface)
void	remove_tracker(tracker: XRTracker)

Signals

interface_added(interface_name: StringName)

Emitted when a new interface has been added.

interface_removed(interface_name: StringName)

Emitted when an interface is removed.

reference_frame_changed()

Emitted when the reference frame transform changes.

tracker_added(tracker_name: StringName, type: int)

Emitted when a new tracker has been added. If you don't use a fixed number of controllers or if you're using XRAnchor3Ds for an AR solution, it is important to react to this signal to add the appropriate XRController3D or XRAnchor3D nodes related to this new tracker.

tracker_removed(tracker_name: StringName, type: int)

Emitted when a tracker is removed. You should remove any XRController3D or XRAnchor3D points if applicable. This is not mandatory, the nodes simply become inactive and will be made active again when a new tracker becomes available (i.e. a new controller is switched on that takes the place of the previous one).

---

tracker_updated(tracker_name: StringName, type: int) 🔗

Emitted when an existing tracker has been updated. This can happen if the user switches controllers.

Enumerations

enum **TrackerType**: 🔗

TrackerType **TRACKER_HEAD** = `1`

The tracker tracks the location of the players head. This is usually a location centered between the players eyes. Note that for handheld AR devices this can be the current location of the device.

TrackerType **TRACKER_CONTROLLER** = `2`

The tracker tracks the location of a controller.

TrackerType **TRACKER_BASESTATION** = `4`

The tracker tracks the location of a base station.

TrackerType **TRACKER_ANCHOR** = `8`

The tracker tracks the location and size of an AR anchor.

TrackerType **TRACKER_HAND** = `16`

The tracker tracks the location and joints of a hand.

TrackerType **TRACKER_BODY** = `32`

The tracker tracks the location and joints of a body.

TrackerType **TRACKER_FACE** = `64`

The tracker tracks the expressions of a face.

TrackerType **TRACKER_ANY_KNOWN** = `127`

Used internally to filter trackers of any known type.

TrackerType **TRACKER_UNKNOWN** = `128`

Used internally if we haven't set the tracker type yet.

TrackerType **TRACKER_ANY** = `255`

Used internally to select all trackers.

enum **RotationMode**: 🔗

RotationMode **RESET_FULL_ROTATION** = `0`

Fully reset the orientation of the HMD. Regardless of what direction the user is looking to in the real world. The user will look dead ahead in the virtual world.

RotationMode **RESET_BUT_KEEP_TILT** = `1`

Resets the orientation but keeps the tilt of the device. So if we're looking down, we keep looking down but heading will be reset.

RotationMode **DONT_RESET_ROTATION** = `2`

Does not reset the orientation of the HMD, only the position of the player gets centered.

Property Descriptions

XRInterface **primary_interface** 🔗

- void **set_primary_interface**(value: XRInterface)
- XRInterface **get_primary_interface**()

The primary XRInterface currently bound to the **XRServer**.

Transform3D **world_origin** = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)` 🔗

- void **set_world_origin**(value: Transform3D)
- Transform3D **get_world_origin**()

The current origin of our tracking space in the virtual world. This is used by the renderer to properly position the camera with new tracking data.

Note: This property is managed by the current XROrigin3D node. It is exposed for access from GDExtensions.

---

float **world_scale** = `1.0` 🔗

- void **set_world_scale**(value: float)
- float **get_world_scale**()

The scale of the game world compared to the real world. By default, most AR/VR platforms assume that 1 game unit corresponds to 1 real world meter.

Method Descriptions

void **add_interface**(interface: XRInterface)

Registers an XRInterface object.

---

void **add_tracker**(tracker: XRTracker)

Registers a new XRTracker that tracks a physical object.

---

void **center_on_hmd**(rotation_mode: RotationMode, keep_height: bool)

This is an important function to understand correctly. AR and VR platforms all handle positioning slightly differently.

For platforms that do not offer spatial tracking, our origin point (0, 0, 0) is the location of our HMD, but you have little control over the direction the player is facing in the real world.

For platforms that do offer spatial tracking, our origin point depends very much on the system. For OpenVR, our origin point is usually the center of the tracking space, on the ground. For other platforms, it's often the location of the tracking camera.

This method allows you to center your tracker on the location of the HMD. It will take the current location of the HMD and use that to adjust all your tracking data; in essence, realigning the real world to your player's current position in the game world.

For this method to produce usable results, tracking information must be available. This often takes a few frames after starting your game.

You should call this method after a few seconds have passed. For example, when the user requests a realignment of the display holding a designated button on a controller for a short period of time, or when implementing a teleport mechanism.

---

void **clear_reference_frame**() 🔗

Clears the reference frame that was set by previous calls to center_on_hmd.

XRInterface **find_interface**(name: String) const 🔗

Finds an interface by its `name`. For example, if your project uses capabilities of an AR/VR platform, you can find the interface for that platform by name and initialize it.

Transform3D **get_hmd_transform**() 🔗

Returns the primary interface's transformation.

XRInterface **get_interface**(idx: int) const 🔗

Returns the interface registered at the given `idx` index in the list of interfaces.

---

int **get_interface_count**() const 🔗

Returns the number of interfaces currently registered with the AR/VR server. If your project supports multiple AR/VR platforms, you can look through the available interface, and either present the user with a selection or simply try to initialize each interface and use the first one that returns `true`.

Array[Dictionary] **get_interfaces**() const

Returns a list of available interfaces the ID and name of each interface.

Transform3D **get_reference_frame**() const

Returns the reference frame transform. Mostly used internally and exposed for GDExtension build interfaces.

XRTracker **get_tracker**(tracker_name: StringName) const

Returns the positional tracker with the given `tracker_name`.

Dictionary **get_trackers**(tracker_types: int)

Returns a dictionary of trackers for `tracker_types`.

---

void **remove_interface**(interface: XRInterface)

Removes this `interface`.

---

void **remove_tracker**(tracker: XRTracker)

Removes this `tracker`.

XRTracker

Inherits: RefCounted **<** Object

Inherited By: XRFaceTracker, XRPositionalTracker

A tracked object.

Description

This object is the base of all XR trackers.

Tutorials

- XR documentation index

Properties

String	description	`""`
StringName	name	`&"Unknown"`
TrackerType	type	`128`

Property Descriptions

String **description** = `""`

- void **set_tracker_desc**(value: String)
- String **get_tracker_desc**()

The description of this tracker.

StringName **name** = `&"Unknown"`

- void **set_tracker_name**(value: StringName)
- StringName **get_tracker_name**()

The unique name of this tracker. The trackers that are available differ between various XR runtimes and can often be configured by the user. Godot maintains a number of reserved names that it expects the XRInterface to implement if applicable:

- `head` identifies the XRPositionalTracker of the players head
- `left_hand` identifies the XRControllerTracker in the players left hand
- `right_hand` identifies the XRControllerTracker in the players right hand
- `/user/hand_tracker/left` identifies the XRHandTracker for the players left hand
- `/user/hand_tracker/right` identifies the XRHandTracker for the players right hand
- `/user/body_tracker` identifies the XRBodyTracker for the players body
- `/user/face_tracker` identifies the XRFaceTracker for the players face

TrackerType **type** = `128`

- void **set_tracker_type**(value: TrackerType)

- TrackerType **get_tracker_type**()

The type of tracker.

XRVRS

Inherits: Object

Helper class for XR interfaces that generates VRS images.

Description

This class is used by various XR interfaces to generate VRS textures that can be used to speed up rendering.

Properties

float	vrs_min_radius	`20.0`
float	vrs_strength	`1.0`

Methods

RID	make_vrs_texture(target_size: Vector2, eye_foci: PackedVector2Array)

Property Descriptions

float **vrs_min_radius** = `20.0`

- void **set_vrs_min_radius**(value: float)
- float **get_vrs_min_radius**()

The minimum radius around the focal point where full quality is guaranteed if VRS is used as a percentage of screen size.

float **vrs_strength** = `1.0`

- void **set_vrs_strength**(value: float)
- float **get_vrs_strength**()

The strength used to calculate the VRS density map. The greater this value, the more noticeable VRS is.

Method Descriptions

RID **make_vrs_texture**(target_size: Vector2, eye_foci: PackedVector2Array)

Generates the VRS texture based on a render `target_size` adjusted by our VRS tile size. For each eyes focal point passed in `eye_foci` a layer is created. Focal point should be in NDC.

The result will be cached, requesting a VRS texture with unchanged parameters and settings will return the cached RID.

ZIPPacker

Inherits: RefCounted **<** Object

Allows the creation of zip files.

Description

This class implements a writer that allows storing the multiple blobs in a zip archive.

```
func write_zip_file():
    var writer := ZIPPacker.new()
    var err := writer.open("user://archive.zip")
    if err != OK:
        return err
    writer.start_file("hello.txt")
    writer.write_file("Hello World".to_utf8_buffer())
    writer.close_file()

    writer.close()
    return OK
```

Methods

Error close()

Error close_file()

Error open(path: String, append: ZipAppend = 0)

Error	start_file(path: String)
Error	write_file(data: PackedByteArray)

Enumerations

enum **ZipAppend**:

ZipAppend **APPEND_CREATE** = `0`

Create a new zip archive at the given path.

ZipAppend **APPEND_CREATEAFTER** = `1`

Append a new zip archive to the end of the already existing file at the given path.

ZipAppend **APPEND_ADDINZIP** = `2`

Add new files to the existing zip archive at the given path.

Method Descriptions

Error **close**()

Closes the underlying resources used by this instance.

Error **close_file**()

Stops writing to a file within the archive.

It will fail if there is no open file.

Error **open**(path: String, append: ZipAppend = 0)

Opens a zip file for writing at the given path using the specified write mode.

This must be called before everything else.

Error **start_file**(path: String)

Starts writing to a file within the archive. Only one file can be written at the same time.

Must be called after open.

Error **write_file**(data: PackedByteArray)

Write the given `data` to the file.

Needs to be called after start_file.

ZIPReader

Inherits: RefCounted **<** Object

Allows reading the content of a zip file.

Description

This class implements a reader that can extract the content of individual files inside a zip archive.

```
func read_zip_file():
    var reader := ZIPReader.new()
    var err := reader.open("user://archive.zip")
    if err != OK:
        return PackedByteArray()
    var res := reader.read_file("hello.txt")
    reader.close()
    return res
```

Methods

Error	close()
bool	file_exists(path: String, case_sensitive: bool = true)
PackedStringArray	get_files()
Error	open(path: String)

PackedByteArray	read_file(path: String, case_sensitive: bool = true)

Method Descriptions

Error **close**()

Closes the underlying resources used by this instance.

bool **file_exists**(path: String, case_sensitive: bool = true)

Returns `true` if the file exists in the loaded zip archive.

Must be called after open.

PackedStringArray **get_files**()

Returns the list of names of all files in the loaded archive.

Must be called after open.

Error **open**(path: String)

Opens the zip archive at the given `path` and reads its file index.

PackedByteArray **read_file**(path: String, case_sensitive: bool = true)

Loads the whole content of a file in the loaded zip archive into memory and returns it.

Must be called after open.

Variant

The most important data type in Godot.

Description

In computer programming, a Variant class is a class that is designed to store a variety of other types. Dynamic programming languages like PHP, Lua, JavaScript and GDScript like to use them to store variables' data on the backend. With these Variants, properties are able to change value types freely.

GDScript

```
var foo = 2 # foo is dynamically an integer
foo = "Now foo is a string!"
foo = RefCounted.new() # foo is an Object
var bar: int = 2 # bar is a statically typed integer.
# bar = "Uh oh! I can't make statically typed variables become a different type!"
```

C#

```
// C# is statically typed. Once a variable has a type it cannot be changed. You can use the `var` keyword to let the compiler infer the type automatically.
var foo = 2; // Foo is a 32-bit integer (int). Be cautious, integers in GDScript are 64-bit and the direct C# equivalent is `long`.
// foo = "foo was and will always be an integer. It cannot be turned into a string!";
var boo = "Boo is a string!";
var ref = new RefCounted(); // var is especially useful when used together with a constructor.

// Godot also provides a Variant type that works like a union
```

```
of all the Variant-compatible types.
Variant fooVar = 2; // fooVar is dynamically an integer (stored
as a `long` in the Variant type).
fooVar = "Now fooVar is a string!";
fooVar = new RefCounted(); // fooVar is a GodotObject.
```

Godot tracks all scripting API variables within Variants. Without even realizing it, you use Variants all the time. When a particular language enforces its own rules for keeping data typed, then that language is applying its own custom logic over the base Variant scripting API.

- GDScript automatically wrap values in them. It keeps all data in plain Variants by default and then optionally enforces custom static typing rules on variable types.
- C# is statically typed, but uses its own implementation of the Variant type in place of Godot's **Variant** class when it needs to represent a dynamic value. C# Variant can be assigned any compatible type implicitly but converting requires an explicit cast.

The global @GlobalScope.typeof function returns the enumerated value of the Variant type stored in the current variable (see Variant.Type).

GDScript

```
var foo = 2
match typeof(foo):
    TYPE_NIL:
        print("foo is null")
    TYPE_INT:
        print("foo is an integer")
    TYPE_OBJECT:
        # Note that Objects are their own special category.
        # To get the name of the underlying Object type, you
need the `get_class()` method.
        print("foo is a(n) %s" % foo.get_class()) # inject the
class name into a formatted string.
        # Note that this does not get the script's `class_name`
```

```
global identifier.
        # If the `class_name` is needed, use
`foo.get_script().get_global_name()` instead.
```

C#

```
Variant foo = 2;
switch (foo.VariantType)
{
    case Variant.Type.Nil:
        GD.Print("foo is null");
        break;
    case Variant.Type.Int:
        GD.Print("foo is an integer");
        break;
    case Variant.Type.Object:
        // Note that Objects are their own special category.
        // You can convert a Variant to a GodotObject and use
reflection to get its name.
        GD.Print($"foo is a(n)
{foo.AsGodotObject().GetType().Name}");
        break;
}
```

A Variant takes up only 20 bytes and can store almost any engine datatype inside of it. Variants are rarely used to hold information for long periods of time. Instead, they are used mainly for communication, editing, serialization and moving data around.

Godot has specifically invested in making its Variant class as flexible as possible; so much so that it is used for a multitude of operations to facilitate communication between all of Godot's systems.

A Variant:

- Can store almost any datatype.
- Can perform operations between many variants. GDScript uses Variant as its atomic/native datatype.
- Can be hashed, so it can be compared quickly to other variants.

- Can be used to convert safely between datatypes.
- Can be used to abstract calling methods and their arguments. Godot exports all its functions through variants.
- Can be used to defer calls or move data between threads.
- Can be serialized as binary and stored to disk, or transferred via network.
- Can be serialized to text and use it for printing values and editable settings.
- Can work as an exported property, so the editor can edit it universally.
- Can be used for dictionaries, arrays, parsers, etc.

Containers (Array and Dictionary): Both are implemented using variants. A Dictionary can match any datatype used as key to any other datatype. An Array just holds an array of Variants. Of course, a Variant can also hold a Dictionary and an Array inside, making it even more flexible.

Modifications to a container will modify all references to it. A Mutex should be created to lock it if multi-threaded access is desired.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Variant class introduction

AABB

A 3D axis-aligned bounding box.

Description

The **AABB** built-in Variant type represents an axis-aligned bounding box in a 3D space. It is defined by its position and size, which are Vector3. It is frequently used for fast overlap tests (see intersects). Although **AABB** itself is axis-aligned, it can be combined with Transform3D to represent a rotated or skewed bounding box.

It uses floating-point coordinates. The 2D counterpart to **AABB** is Rect2. There is no version of **AABB** that uses integer coordinates.

Note: Negative values for size are not supported. With negative size, most **AABB** methods do not work correctly. Use abs to get an equivalent **AABB** with a non-negative size.

Note: In a boolean context, a **AABB** evaluates to `false` if both position and size are zero (equal to Vector3.ZERO). Otherwise, it always evaluates to `true`.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Math documentation index

- Vector math
- Advanced vector math

Properties

Vector3	end	`Vector3(0, 0, 0)`
Vector3	position	`Vector3(0, 0, 0)`
Vector3	size	`Vector3(0, 0, 0)`

Constructors

AABB	AABB()
AABB	AABB(from: AABB)
AABB	AABB(position: Vector3, size: Vector3)

Methods

AABB	abs() const
bool	encloses(with: AABB) const

AABB	expand(to_point: Vector3) const
Vector3	get_center() const
Vector3	get_endpoint(idx: int) const
Vector3	get_longest_axis() const
int	get_longest_axis_index() const
float	get_longest_axis_size() const
Vector3	get_shortest_axis() const
int	get_shortest_axis_index() const
float	get_shortest_axis_size() const
Vector3	get_support(dir: Vector3) const
float	get_volume() const
AABB	grow(by: float) const
bool	has_point(point: Vector3) const

bool	has_surface() const
bool	has_volume() const
AABB	intersection(with: AABB) const
bool	intersects(with: AABB) const
bool	intersects_plane(plane: Plane) const
Variant	intersects_ray(from: Vector3, dir: Vector3) const
Variant	intersects_segment(from: Vector3, to: Vector3) const
bool	is_equal_approx(aabb: AABB) const
bool	is_finite() const
AABB	merge(with: AABB) const

Operators

bool	operator !=(right: AABB)

AABB	operator *(right: Transform3D)
bool	operator ==(right: AABB)

Property Descriptions

Vector3 **end** = `Vector3(0, 0, 0)` 🔗

The ending point. This is usually the corner on the top-right and forward of the bounding box, and is equivalent to `position + size`. Setting this point affects the size.

Vector3 **position** = `Vector3(0, 0, 0)` 🔗

The origin point. This is usually the corner on the bottom-left and back of the bounding box.

Vector3 **size** = `Vector3(0, 0, 0)` 🔗

The bounding box's width, height, and depth starting from position. Setting this value also affects the end point.

Note: It's recommended setting the width, height, and depth to non-negative values. This is because most methods in Godot assume that the position is the bottom-left-back corner, and the end is the top-right-forward corner. To get an equivalent bounding box with non-negative size, use abs.

Constructor Descriptions

AABB **AABB**() 🔗

Constructs an **AABB** with its position and size set to Vector3.ZERO.

AABB **AABB**(from: AABB)

Constructs an **AABB** as a copy of the given **AABB**.

AABB **AABB**(position: Vector3, size: Vector3)

Constructs an **AABB** by `position` and `size`.

Method Descriptions

AABB **abs**() const 🔗

Returns an **AABB** equivalent to this bounding box, with its width, height, and depth modified to be non-negative values.

GDScript

```
var box = AABB(Vector3(5, 0, 5), Vector3(-20, -10, -5))
var absolute = box.abs()
print(absolute.position) # Prints (-15, -10, 0)
print(absolute.size)     # Prints (20, 10, 5)
```

C#

```
var box = new Aabb(new Vector3(5, 0, 5), new Vector3(-20, -10,
-5));
var absolute = box.Abs();
```

```
GD.Print(absolute.Position); // Prints (-15, -10, 0)
GD.Print(absolute.Size);     // Prints (20, 10, 5)
```

Note: It's recommended to use this method when size is negative, as most other methods in Godot assume that the size's components are greater than `0`.

---

bool **encloses**(with: AABB) const

Returns `true` if this bounding box *completely* encloses the `with` box. The edges of both boxes are included.

GDScript

```
var a = AABB(Vector3(0, 0, 0), Vector3(4, 4, 4))
var b = AABB(Vector3(1, 1, 1), Vector3(3, 3, 3))
var c = AABB(Vector3(2, 2, 2), Vector3(8, 8, 8))

print(a.encloses(a)) # Prints true
print(a.encloses(b)) # Prints true
print(a.encloses(c)) # Prints false
```

C#

```
var a = new Aabb(new Vector3(0, 0, 0), new Vector3(4, 4, 4));
var b = new Aabb(new Vector3(1, 1, 1), new Vector3(3, 3, 3));
var c = new Aabb(new Vector3(2, 2, 2), new Vector3(8, 8, 8));

GD.Print(a.Encloses(a)); // Prints True
GD.Print(a.Encloses(b)); // Prints True
GD.Print(a.Encloses(c)); // Prints False
```

AABB **expand**(to_point: Vector3) const

Returns a copy of this bounding box expanded to align the edges with the given `to_point`, if necessary.

GDScript

```
var box = AABB(Vector3(0, 0, 0), Vector3(5, 2, 5))

box = box.expand(Vector3(10, 0, 0))
print(box.position) # Prints (0, 0, 0)
print(box.size)     # Prints (10, 2, 5)

box = box.expand(Vector3(-5, 0, 5))
print(box.position) # Prints (-5, 0, 0)
print(box.size)     # Prints (15, 2, 5)
```

C#

```
var box = new Aabb(new Vector3(0, 0, 0), new Vector3(5, 2, 5));

box = box.Expand(new Vector3(10, 0, 0));
GD.Print(box.Position); // Prints (0, 0, 0)
GD.Print(box.Size);     // Prints (10, 2, 5)

box = box.Expand(new Vector3(-5, 0, 5));
GD.Print(box.Position); // Prints (-5, 0, 0)
GD.Print(box.Size);     // Prints (15, 2, 5)
```

Vector3 **get_center**() const

Returns the center point of the bounding box. This is the same as `position + (size / 2.0)`.

Vector3 **get_endpoint**(idx: int) const

Returns the position of one of the 8 vertices that compose this bounding box. With a `idx` of `0` this is the same as position, and a `idx` of `7` is the same as end.

Vector3 **get_longest_axis**() const

Returns the longest normalized axis of this bounding box's size, as a Vector3 (Vector3.RIGHT, Vector3.UP, or Vector3.BACK).

GDScript

```
var box = AABB(Vector3(0, 0, 0), Vector3(2, 4, 8))

print(box.get_longest_axis())       # Prints (0, 0, 1)
print(box.get_longest_axis_index()) # Prints 2
print(box.get_longest_axis_size())  # Prints 8
```

C#

```
var box = new Aabb(new Vector3(0, 0, 0), new Vector3(2, 4, 8));

GD.Print(box.GetLongestAxis());      // Prints (0, 0, 1)
GD.Print(box.GetLongestAxisIndex()); // Prints 2
GD.Print(box.GetLongestAxisSize());  // Prints 8
```

See also get_longest_axis_index and get_longest_axis_size.

---

int **get_longest_axis_index**() const

Returns the index to the longest axis of this bounding box's size (see Vector3.AXIS_X, Vector3.AXIS_Y, and Vector3.AXIS_Z).

For an example, see get_longest_axis.

---

float **get_longest_axis_size**() const

Returns the longest dimension of this bounding box's size.

For an example, see get_longest_axis.

Vector3 **get_shortest_axis**() const

Returns the shortest normalized axis of this bounding box's size, as a Vector3 (Vector3.RIGHT, Vector3.UP, or Vector3.BACK).

GDScript

```
var box = AABB(Vector3(0, 0, 0), Vector3(2, 4, 8))

print(box.get_shortest_axis())       # Prints (1, 0, 0)
print(box.get_shortest_axis_index()) # Prints 0
print(box.get_shortest_axis_size())  # Prints 2
```

C#

```
var box = new Aabb(new Vector3(0, 0, 0), new Vector3(2, 4, 8));

GD.Print(box.GetShortestAxis());      // Prints (1, 0, 0)
GD.Print(box.GetShortestAxisIndex()); // Prints 0
GD.Print(box.GetShortestAxisSize());  // Prints 2
```

See also get_shortest_axis_index and get_shortest_axis_size.

---

int **get_shortest_axis_index**() const 🔗

Returns the index to the shortest axis of this bounding box's size (see Vector3.AXIS_X, Vector3.AXIS_Y, and Vector3.AXIS_Z).

For an example, see get_shortest_axis.

---

float **get_shortest_axis_size**() const 🔗

Returns the shortest dimension of this bounding box's size.

For an example, see get_shortest_axis.

Vector3 **get_support**(dir: Vector3) const 🔗

Returns the vertex's position of this bounding box that's the farthest in the given direction. This point is commonly known as the support point in collision detection algorithms.

---

float **get_volume**() const

Returns the bounding box's volume. This is equivalent to `size.x * size.y * size.z`. See also has_volume.

AABB **grow**(by: float) const

Returns a copy of this bounding box extended on all sides by the given amount `by`. A negative amount shrinks the box instead.

GDScript

```
var a = AABB(Vector3(4, 4, 4), Vector3(8, 8, 8)).grow(4)
print(a.position) # Prints (0, 0, 0)
print(a.size)      # Prints (16, 16, 16)

var b = AABB(Vector3(0, 0, 0), Vector3(8, 4, 2)).grow(2)
print(b.position) # Prints (-2, -2, -2)
print(b.size)      # Prints (12, 8, 6)
```

C#

```
var a = new Aabb(new Vector3(4, 4, 4), new Vector3(8, 8,
8)).Grow(4);
GD.Print(a.Position); // Prints (0, 0, 0)
GD.Print(a.Size);      // Prints (16, 16, 16)

var b = new Aabb(new Vector3(0, 0, 0), new Vector3(8, 4,
2)).Grow(2);
GD.Print(b.Position); // Prints (-2, -2, -2)
GD.Print(b.Size);      // Prints (12, 8, 6)
```

---

bool **has_point**(point: Vector3) const

Returns `true` if the bounding box contains the given `point`. By convention, points exactly on the right, top, and front sides are **not** included.

Note: This method is not reliable for **AABB** with a *negative* size. Use abs first to get a valid bounding box.

---

bool **has_surface**() const

Returns `true` if this bounding box has a surface or a length, that is, at least one component of size is greater than `0`. Otherwise, returns `false`.

---

bool **has_volume**() const

Returns `true` if this bounding box's width, height, and depth are all positive. See also get_volume.

AABB **intersection**(with: AABB) const

Returns the intersection between this bounding box and `with`. If the boxes do not intersect, returns an empty **AABB**. If the boxes intersect at the edge, returns a flat **AABB** with no volume (see has_surface and has_volume).

GDScript

```
var box1 = AABB(Vector3(0, 0, 0), Vector3(5, 2, 8))
var box2 = AABB(Vector3(2, 0, 2), Vector3(8, 4, 4))

var intersection = box1.intersection(box2)
```

```
print(intersection.position) # Prints (2, 0, 2)
print(intersection.size)     # Prints (3, 2, 4)
```

C#

```
var box1 = new Aabb(new Vector3(0, 0, 0), new Vector3(5, 2,
8));
var box2 = new Aabb(new Vector3(2, 0, 2), new Vector3(8, 4,
4));

var intersection = box1.Intersection(box2);
GD.Print(intersection.Position); // Prints (2, 0, 2)
GD.Print(intersection.Size);     // Prints (3, 2, 4)
```

Note: If you only need to know whether two bounding boxes are intersecting, use intersects, instead.

---

bool **intersects**(with: AABB) const

Returns `true` if this bounding box overlaps with the box `with`. The edges of both boxes are *always* excluded.

---

bool **intersects_plane**(plane: Plane) const

Returns `true` if this bounding box is on both sides of the given `plane`.

Variant **intersects_ray**(from: Vector3, dir: Vector3) const

Returns the first point where this bounding box and the given ray intersect, as a Vector3. If no intersection occurs, returns `null`.

The ray begin at `from`, faces `dir` and extends towards infinity.

Variant **intersects_segment**(from: Vector3, to: Vector3) const

Returns the first point where this bounding box and the given segment intersect, as a Vector3. If no intersection occurs, returns `null`.

The segment begins at `from` and ends at `to`.

---

bool **is_equal_approx**(aabb: AABB) const

Returns `true` if this bounding box and `aabb` are approximately equal, by calling Vector2.is_equal_approx on the position and the size.

---

bool **is_finite**() const

Returns `true` if this bounding box's values are finite, by calling Vector2.is_finite on the position and the size.

AABB **merge**(with: AABB) const

Returns an **AABB** that encloses both this bounding box and `with` around the edges. See also encloses.

Operator Descriptions

bool **operator !=**(right: AABB)

Returns `true` if the position or size of both bounding boxes are not equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

AABB **operator ***(right: Transform3D)

Inversely transforms (multiplies) the **AABB** by the given Transform3D transformation matrix, under the assumption that the transformation basis is orthonormal (i.e. rotation/reflection is fine, scaling/skew is not).

`aabb * transform` is equivalent to `transform.inverse() * aabb`. See Transform3D.inverse.

For transforming by inverse of an affine transformation (e.g. with scaling) `transform.affine_inverse() * aabb` can be used instead. See Transform3D.affine_inverse.

---

bool **operator ==**(right: AABB)

Returns `true` if both position and size of the bounding boxes are exactly equal, respectively.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Array

A built-in data structure that holds a sequence of elements.

Description

An array data structure that can contain a sequence of elements of any Variant type. Elements are accessed by a numerical index starting at 0. Negative indices are used to count from the back (-1 is the last element, -2 is the second to last, etc.).

Example:

GDScript

```
var array = ["First", 2, 3, "Last"]
print(array[0])  # Prints "First"
print(array[2])  # Prints 3
print(array[-1]) # Prints "Last"

array[1] = "Second"
print(array[1])  # Prints "Second"
print(array[-3]) # Prints "Second"
```

C#

```
var array = new Godot.Collections.Array{"First", 2, 3, "Last"};
GD.Print(array[0]); // Prints "First"
GD.Print(array[2]); // Prints 3
GD.Print(array[array.Count - 1]); // Prints "Last"

array[2] = "Second";
GD.Print(array[1]); // Prints "Second"
GD.Print(array[array.Count - 3]); // Prints "Second"
```

Note: Arrays are always passed by **reference**. To get a copy of an array that can be modified independently of the original array, use duplicate.

Note: Erasing elements while iterating over arrays is **not** supported and will result in unpredictable behavior.

Differences between packed arrays, typed arrays, and untyped arrays: Packed arrays are generally faster to iterate on and modify compared to a typed array of the same type (e.g. PackedInt64Array versus `Array[int]`). Also, packed arrays consume less memory. As a downside, packed arrays are less flexible as they don't offer as many convenience methods such as map. Typed arrays are in turn faster to iterate on and modify than untyped arrays.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

Array	Array()
Array	Array(base: Array, type: int, class_name: StringName, script: Variant)
Array	Array(from: Array)
Array	Array(from: PackedByteArray)

Array	Array(from: PackedColorArray)
Array	Array(from: PackedFloat32Array)
Array	Array(from: PackedFloat64Array)
Array	Array(from: PackedInt32Array)
Array	Array(from: PackedInt64Array)
Array	Array(from: PackedStringArray)
Array	Array(from: PackedVector2Array)
Array	Array(from: PackedVector3Array)
Array	Array(from: PackedVector4Array)

Methods

bool	all(method: Callable) const
bool	any(method: Callable) const

void	append(value: Variant)
void	append_array(array: Array)
void	assign(array: Array)
Variant	back() const
int	bsearch(value: Variant, before: bool = true) const
int	bsearch_custom(value: Variant, func: Callable, before: bool = true) const
void	clear()
int	count(value: Variant) const
Array	duplicate(deep: bool = false) const
void	erase(value: Variant)
void	fill(value: Variant)
Array	filter(method: Callable) const

int	find(what: Variant, from: int = 0) const
Variant	front() const
int	get_typed_builtin() const
StringName	get_typed_class_name() const
Variant	get_typed_script() const
bool	has(value: Variant) const
int	hash() const
int	insert(position: int, value: Variant)
bool	is_empty() const
bool	is_read_only() const
bool	is_same_typed(array: Array) const
bool	is_typed() const
void	make_read_only()

Array	map(method: Callable) const
Variant	max() const
Variant	min() const
Variant	pick_random() const
Variant	pop_at(position: int)
Variant	pop_back()
Variant	pop_front()
void	push_back(value: Variant)
void	push_front(value: Variant)
Variant	reduce(method: Callable, accum: Variant = null) const
void	remove_at(position: int)
int	resize(size: int)
void	reverse()

int	rfind(what: Variant, from: int = -1) const
void	shuffle()
int	size() const
Array	slice(begin: int, end: int = 2147483647, step: int = 1, deep: bool = false) const
void	sort()
void	sort_custom(func: Callable)

Operators

bool	operator !=(right: Array)
Array	operator +(right: Array)
bool	operator <(right: Array)
bool	operator <=(right: Array)
bool	operator ==(right: Array)

bool	operator >(right: Array)
bool	operator >=(right: Array)
Variant	operator [](index: int)

Constructor Descriptions

Array **Array**() 🔗

Constructs an empty **Array**.

Array **Array**(base: Array, type: int, class_name: StringName, script: Variant)

Creates a typed array from the `base` array. A typed array can only contain elements of the given type, or that inherit from the given class, as described by this constructor's parameters:

- `type` is the built-in Variant type, as one the Variant.Type constants.
- `class_name` is the built-in class name (see Object.get_class).
- `script` is the associated script. It must be a Script instance or `null`.

If `type` is not @GlobalScope.TYPE_OBJECT, `class_name` must be an empty StringName and `script` must be `null`.

```
class_name Sword
extends Node
```

```
class Stats:
    pass

func _ready():
    var a = Array([], TYPE_INT, "", null)               #
Array[int]
    var b = Array([], TYPE_OBJECT, "Node", null)        #
Array[Node]
    var c = Array([], TYPE_OBJECT, "Node", Sword)       #
Array[Sword]
    var d = Array([], TYPE_OBJECT, "RefCounted", Stats) #
Array[Stats]
```

The `base` array's elements are converted when necessary. If this is not possible or `base` is already typed, this constructor fails and returns an empty **Array**.

In GDScript, this constructor is usually not necessary, as it is possible to create a typed array through static typing:

```
var numbers: Array[float] = []
var children: Array[Node] = [$Node, $Sprite2D, $RigidBody3D]

var integers: Array[int] = [0.2, 4.5, -2.0]
print(integers) # Prints [0, 4, -2]
```

Array **Array**(from: Array)

Returns the same array as `from`. If you need a copy of the array, use duplicate.

Array **Array**(from: PackedByteArray)

Constructs an array from a PackedByteArray.

Array **Array**(from: PackedColorArray)

Constructs an array from a PackedColorArray.

Array **Array**(from: PackedFloat32Array)

Constructs an array from a PackedFloat32Array.

Array **Array**(from: PackedFloat64Array)

Constructs an array from a PackedFloat64Array.

Array **Array**(from: PackedInt32Array)

Constructs an array from a PackedInt32Array.

Array **Array**(from: PackedInt64Array)

Constructs an array from a PackedInt64Array.

Array **Array**(from: PackedStringArray)

Constructs an array from a PackedStringArray.

Array **Array**(from: PackedVector2Array)

Constructs an array from a PackedVector2Array.

Array **Array**(from: PackedVector3Array)

Constructs an array from a PackedVector3Array.

Array **Array**(from: PackedVector4Array)

Constructs an array from a PackedVector4Array.

Method Descriptions

bool **all**(method: Callable) const

Calls the given Callable on each element in the array and returns `true` if the Callable returns `true` for *all* elements in the array. If the Callable returns `false` for one array element or more, this method returns `false`.

The `method` should take one Variant parameter (the current array element) and return a bool.

GDScript

```
func greater_than_5(number):
    return number > 5

func _ready():
    print([6, 10, 6].all(greater_than_5)) # Prints true (3/3 elements evaluate to true).
    print([4, 10, 4].all(greater_than_5)) # Prints false (1/3 elements evaluate to true).
    print([4, 4, 4].all(greater_than_5))  # Prints false (0/3 elements evaluate to true).
    print([].all(greater_than_5))         # Prints true (0/0 elements evaluate to true).

    # Same as the first line above, but using a lambda function.
    print([6, 10, 6].all(func(element): return element > 5)) # Prints true
```

C#

```
private static bool GreaterThan5(int number)
{
    return number > 5;
}

public override void _Ready()
{
    // Prints true (3/3 elements evaluate to true).
    GD.Print(new Godot.Collections.Array>int< { 6, 10, 6
}.All(GreaterThan5));
    // Prints false (1/3 elements evaluate to true).
    GD.Print(new Godot.Collections.Array>int< { 4, 10, 4
}.All(GreaterThan5));
    // Prints false (0/3 elements evaluate to true).
    GD.Print(new Godot.Collections.Array>int< { 4, 4, 4
}.All(GreaterThan5));
    // Prints true (0/0 elements evaluate to true).
    GD.Print(new Godot.Collections.Array>int< {
}.All(GreaterThan5));

    // Same as the first line above, but using a lambda
function.
    GD.Print(new Godot.Collections.Array>int< { 6, 10, 6
}.All(element => element > 5)); // Prints true
}
```

See also any, filter, map and reduce.

Note: Unlike relying on the size of an array returned by filter, this method will return as early as possible to improve performance (especially with large arrays).

Note: For an empty array, this method always [https://en.wikipedia.org/wiki/Vacuous_truth] returns `true`.

---

bool **any**(method: Callable) const

Calls the given Callable on each element in the array and returns `true` if the Callable returns `true` for *one or more* elements in the array. If the Callable returns `false` for all elements in the array, this method returns `false`.

The `method` should take one Variant parameter (the current array element) and return a bool.

```
func greater_than_5(number):
    return number > 5

func _ready():
    print([6, 10, 6].any(greater_than_5)) # Prints true (3
elements evaluate to true).
    print([4, 10, 4].any(greater_than_5)) # Prints true (1
elements evaluate to true).
    print([4, 4, 4].any(greater_than_5))  # Prints false (0
elements evaluate to true).
    print([].any(greater_than_5))         # Prints false (0
elements evaluate to true).

    # Same as the first line above, but using a lambda
function.
    print([6, 10, 6].any(func(number): return number > 5))  #
Prints true
```

See also all, filter, map and reduce.

Note: Unlike relying on the size of an array returned by filter, this method will return as early as possible to improve performance (especially with large arrays).

Note: For an empty array, this method always returns `false`.

void **append**(value: Variant)

Appends `value` at the end of the array (alias of push_back).

void **append_array**(array: Array)

Appends another `array` at the end of this array.

```
var numbers = [1, 2, 3]
var extra = [4, 5, 6]
numbers.append_array(extra)
print(numbers) # Prints [1, 2, 3, 4, 5, 6]
```

void **assign**(array: Array)

Assigns elements of another `array` into the array. Resizes the array to match `array`. Performs type conversions if the array is typed.

Variant **back**() const

Returns the last element of the array. If the array is empty, fails and returns `null`. See also front.

Note: Unlike with the `[]` operator (`array[-1]`), an error is generated without stopping project execution.

int **bsearch**(value: Variant, before: bool = true) const

Returns the index of `value` in the sorted array. If it cannot be found, returns where `value` should be inserted to keep the array sorted. The algorithm used is binary search [https://en.wikipedia.org/wiki/Binary_search_algorithm].

If `before` is `true` (as by default), the returned index comes before all existing elements equal to `value` in the array.

```
var numbers = [2, 4, 8, 10]
var idx = numbers.bsearch(7)
```

```
numbers.insert(idx, 7)
print(numbers) # Prints [2, 4, 7, 8, 10]

var fruits = ["Apple", "Lemon", "Lemon", "Orange"]
print(fruits.bsearch("Lemon", true))  # Prints 1, points at the
first "Lemon".
print(fruits.bsearch("Lemon", false)) # Prints 3, points at
"Orange".
```

Note: Calling bsearch on an *unsorted* array will result in unexpected behavior. Use sort before calling this method.

---

int **bsearch_custom**(value: Variant, func: Callable, before: bool = true) const

Returns the index of `value` in the sorted array. If it cannot be found, returns where `value` should be inserted to keep the array sorted (using `func` for the comparisons). The algorithm used is binary search [https://en.wikipedia.org/wiki/Binary_search_algorithm].

Similar to sort_custom, `func` is called as many times as necessary, receiving one array element and `value` as arguments. The function should return `true` if the array element should be *behind* `value`, otherwise it should return `false`.

If `before` is `true` (as by default), the returned index comes before all existing elements equal to `value` in the array.

```
func sort_by_amount(a, b):
    if a[1] < b[1]:
        return true
    return false

func _ready():
    var my_items = [["Tomato", 2], ["Kiwi", 5], ["Rice", 9]]

    var apple = ["Apple", 5]
    # "Apple" is inserted before "Kiwi".
    my_items.insert(my_items.bsearch_custom(apple,
sort_by_amount, true), apple)

    var banana = ["Banana", 5]
    # "Banana" is inserted after "Kiwi".
    my_items.insert(my_items.bsearch_custom(banana,
sort_by_amount, false), banana)

    # Prints [["Tomato", 2], ["Apple", 5], ["Kiwi", 5],
["Banana", 5], ["Rice", 9]]
    print(my_items)
```

Note: Calling bsearch_custom on an *unsorted* array will result in unexpected behavior. Use sort_custom with `func` before calling this method.

---

void **clear**() 🔗

Removes all elements from the array. This is equivalent to using resize with a size of `0`.

---

int **count**(value: Variant) const 🔗

Returns the number of times an element is in the array.

Array **duplicate**(deep: bool = false) const 🔗

Returns a new copy of the array.

By default, a **shallow** copy is returned: all nested **Array** and Dictionary elements are shared with the original array. Modifying them in one array will also affect them in the other.

If `deep` is `true`, a **deep** copy is returned: all nested arrays and dictionaries are also duplicated (recursively).

---

void **erase**(value: Variant) 🔗

Finds and removes the first occurrence of `value` from the array. If `value` does not exist in the array, nothing happens. To remove an element by index, use remove_at instead.

Note: This method shifts every element's index after the removed `value` back, which may have a noticeable performance cost, especially on larger arrays.

Note: Erasing elements while iterating over arrays is **not** supported and will result in unpredictable behavior.

---

void **fill**(value: Variant) 🔗

Assigns the given `value` to all elements in the array.

This method can often be combined with resize to create an array with a given size and initialized elements:

GDScript

```
var array = []
array.resize(5)
array.fill(2)
print(array) # Prints [2, 2, 2, 2, 2]
```

C#

```
var array = new Godot.Collections.Array();
array.Resize(5);
array.Fill(2);
GD.Print(array); // Prints [2, 2, 2, 2, 2]
```

Note: If `value` is a Variant passed by reference (Object-derived, **Array**, Dictionary, etc.), the array will be filled with references to the same `value`, which are not duplicates.

Array **filter**(method: Callable) const 🔗

Calls the given Callable on each element in the array and returns a new, filtered **Array**.

The `method` receives one of the array elements as an argument, and should return `true` to add the element to the filtered array, or `false` to exclude it.

```
func is_even(number):
    return number % 2 == 0

func _ready():
    print([1, 4, 5, 8].filter(is_even)) # Prints [4, 8]

    # Same as above, but using a lambda function.
    print([1, 4, 5, 8].filter(func(number): return number % 2 == 0))
```

See also any, all, map and reduce.

---

int **find**(what: Variant, from: int = 0) const 🔗

Returns the index of the **first** occurrence of `what` in this array, or `-1` if there are none. The search's start can be specified with `from`, continuing to the end of the array.

Note: If you just want to know whether the array contains `what`, use has (`Contains` in C#). In GDScript, you may also use the `in` operator.

Note: For performance reasons, the search is affected by `what`'s Variant.Type. For example, `7` (int) and `7.0` (float) are not considered equal for this method.

Variant **front**() const 🔗

Returns the first element of the array. If the array is empty, fails and returns `null`. See also back.

Note: Unlike with the `[]` operator (`array[0]`), an error is generated without stopping project execution.

int **get_typed_builtin**() const 🔗

Returns the built-in Variant type of the typed array as a Variant.Type constant. If the array is not typed, returns @GlobalScope.TYPE_NIL. See also is_typed.

StringName **get_typed_class_name**() const 🔗

Returns the **built-in** class name of the typed array, if the built-in Variant type @GlobalScope.TYPE_OBJECT. Otherwise, returns an empty StringName. See also is_typed and Object.get_class.

Variant **get_typed_script**() const 🔗

Returns the Script instance associated with this typed array, or `null` if it does not exist. See also is_typed.

bool **has**(value: Variant) const 🔗

Returns `true` if the array contains the given `value`.

GDScript

```
print(["inside", 7].has("inside"))   # Prints true
print(["inside", 7].has("outside")) # Prints false
print(["inside", 7].has(7))         # Prints true
print(["inside", 7].has("7"))       # Prints false
```

C#

```
var arr = new Godot.Collections.Array { "inside", 7 };
// By C# convention, this method is renamed to `Contains`.
GD.Print(arr.Contains("inside"));   // Prints true
GD.Print(arr.Contains("outside")); // Prints false
GD.Print(arr.Contains(7));         // Prints true
GD.Print(arr.Contains("7"));       // Prints false
```

In GDScript, this is equivalent to the `in` operator:

```
if 4 in [2, 4, 6, 8]:
    print("4 is here!") # Will be printed.
```

Note: For performance reasons, the search is affected by the `value`'s Variant.Type. For example, `7` (int) and `7.0` (float) are not considered equal for this method.

---

int **hash**() const 🔗

Returns a hashed 32-bit integer value representing the array and its contents.

Note: Arrays with equal hash values are *not* guaranteed to be the same, as a result of hash collisions. On the countrary, arrays with different hash values are guaranteed to be different.

---

int **insert**(position: int, value: Variant) 🔗

Inserts a new element (`value`) at a given index (`position`) in the array. `position` should be between `0` and the array's size.

Returns @GlobalScope.OK on success, or one of the other Error constants if this method fails.

Note: Every element's index after `position` needs to be shifted forward, which may have a noticeable performance cost, especially on larger arrays.

bool **is_empty**() const 🔗

Returns `true` if the array is empty (`[]`). See also size.

bool **is_read_only**() const 🔗

Returns `true` if the array is read-only. See make_read_only.

In GDScript, arrays are automatically read-only if declared with the `const` keyword.

bool **is_same_typed**(array: Array) const 🔗

Returns `true` if this array is typed the same as the given `array`. See also is_typed.

bool **is_typed**() const 🔗

Returns `true` if the array is typed. Typed arrays can only contain elements of a specific type, as defined by the typed array constructor. The methods of a typed array are still expected to return a generic Variant.

In GDScript, it is possible to define a typed array with static typing:

```
var numbers: Array[float] = [0.2, 4.2, -2.0]
print(numbers.is_typed()) # Prints true
```

---

void **make_read_only**()

Makes the array read-only. The array's elements cannot be overridden with different values, and their order cannot change. Does not apply to nested elements, such as dictionaries.

In GDScript, arrays are automatically read-only if declared with the `const` keyword.

Array **map**(method: Callable) const

Calls the given Callable for each element in the array and returns a new array filled with values returned by the `method`.

The `method` should take one Variant parameter (the current array element) and can return any Variant.

```
func double(number):
    return number * 2

func _ready():
    print([1, 2, 3].map(double)) # Prints [2, 4, 6]

    # Same as above, but using a lambda function.
    print([1, 2, 3].map(func(element): return element * 2))
```

See also filter, reduce, any and all.

Variant **max**() const

Returns the maximum value contained in the array, if all elements can be compared. Otherwise, returns `null`. See also min.

To find the maximum value using a custom comparator, you can use reduce.

Variant **min**() const

Returns the minimum value contained in the array, if all elements can be compared. Otherwise, returns `null`. See also max.

Variant **pick_random**() const

Returns a random element from the array. Generates an error and returns `null` if the array is empty.

GDScript

```
# May print 1, 2, 3.25, or "Hi".
print([1, 2, 3.25, "Hi"].pick_random())
```

C#

```
var array = new Godot.Collections.Array { 1, 2, 3.25f, "Hi" };
GD.Print(array.PickRandom()); // May print 1, 2, 3.25, or "Hi".
```

Note: Like many similar functions in the engine (such as @GlobalScope.randi or shuffle), this method uses a common, global random seed. To get a predictable outcome from this method, see @GlobalScope.seed.

Variant **pop_at**(position: int)

Removes and returns the element of the array at index `position`. If negative, `position` is considered relative to the end of the array. Returns `null` if the array is empty. If `position` is out of bounds, an error message is also generated.

Note: This method shifts every element's index after `position` back, which may have a noticeable performance cost, especially on larger arrays.

Variant **pop_back**()

Removes and returns the last element of the array. Returns `null` if the array is empty, without generating an error. See also pop_front.

Variant **pop_front**()

Removes and returns the first element of the array. Returns `null` if the array is empty, without generating an error. See also pop_back.

Note: This method shifts every other element's index back, which may have a noticeable performance cost, especially on larger arrays.

void **push_back**(value: Variant)

Appends an element at the end of the array. See also push_front.

void **push_front**(value: Variant)

Adds an element at the beginning of the array. See also push_back.

Note: This method shifts every other element's index forward, which may have a noticeable performance cost, especially on larger arrays.

Variant **reduce**(method: Callable, accum: Variant = null) const

Calls the given Callable for each element in array, accumulates the result in `accum`, then returns it.

The `method` takes two arguments: the current value of `accum` and the current array element. If `accum` is `null` (as by default), the iteration will start from the second element, with the first one used as initial value of `accum`.

```
func sum(accum, number):
    return accum + number

func _ready():
    print([1, 2, 3].reduce(sum, 0))  # Prints 6
    print([1, 2, 3].reduce(sum, 10)) # Prints 16

    # Same as above, but using a lambda function.
    print([1, 2, 3].reduce(func(accum, number): return accum + 
number, 10))
```

If max is not desirable, this method may also be used to implement a custom comparator:

```
func _ready():
    var arr = [Vector2(5, 0), Vector2(3, 4), Vector2(1, 2)]

    var longest_vec = arr.reduce(func(max, vec): return vec if 
is_length_greater(vec, max) else max)
    print(longest_vec) # Prints Vector2(3, 4).

func is_length_greater(a, b):
    return a.length() > b.length()
```

See also map, filter, any and all.

---

void **remove_at**(position: int)

Removes the element from the array at the given index (`position`). If the index is out of bounds, this method fails.

If you need to return the removed element, use pop_at. To remove an element by value, use erase instead.

Note: This method shifts every element's index after `position` back, which may have a noticeable performance cost, especially on larger arrays.

Note: The `position` cannot be negative. To remove an element relative to the end of the array, use `arr.remove_at(arr.size() - (i + 1))`. To remove the last element from the array, use `arr.resize(arr.size() - 1)`.

---

int **resize**(size: int) 🔗

Sets the array's number of elements to `size`. If `size` is smaller than the array's current size, the elements at the end are removed. If `size` is greater, new default elements (usually `null`) are added, depending on the array's type.

Returns @GlobalScope.OK on success, or one of the other Error constants if this method fails.

Note: Calling this method once and assigning the new values is faster than calling append for every new element.

---

void **reverse**() 🔗

Reverses the order of all elements in the array.

---

int **rfind**(what: Variant, from: int = -1) const 🔗

Returns the index of the **last** occurrence of `what` in this array, or `-1` if there are none. The search's start can be specified with `from`, continuing to the beginning of the array. This method is the reverse of find.

---

void **shuffle**()

Shuffles all elements of the array in a random order.

Note: Like many similar functions in the engine (such as @GlobalScope.randi or pick_random), this method uses a common, global random seed. To get a predictable outcome from this method, see @GlobalScope.seed.

---

int **size**() const

Returns the number of elements in the array. Empty arrays (`[]`) always return `0`. See also is_empty.

Array **slice**(begin: int, end: int = 2147483647, step: int = 1, deep: bool = false) const

Returns a new **Array** containing this array's elements, from index `begin` (inclusive) to `end` (exclusive), every `step` elements.

If either `begin` or `end` are negative, their value is relative to the end of the array.

If `step` is negative, this method iterates through the array in reverse, returning a slice ordered backwards. For this to work, `begin` must be greater than `end`.

If `deep` is `true`, all nested **Array** and Dictionary elements in the slice are duplicated from the original, recursively. See also duplicate).

```
var letters = ["A", "B", "C", "D", "E", "F"]

print(letters.slice(0, 2))  # Prints ["A", "B"]
print(letters.slice(2, -2)) # Prints ["C", "D"]
print(letters.slice(-2, 6)) # Prints ["E", "F"]

print(letters.slice(0, 6, 2))  # Prints ["A", "C", "E"]
print(letters.slice(4, 1, -1)) # Prints ["E", "D", "C"]
```

---

void **sort**()

Sorts the array in ascending order. The final order is dependent on the "less than" (`<`) comparison between elements.

GDScript

```
var numbers = [10, 5, 2.5, 8]
numbers.sort()
print(numbers) # Prints [2.5, 5, 8, 10]
```

C#

```
var numbers = new Godot.Collections.Array { 10, 5, 2.5, 8 };
numbers.Sort();
GD.Print(numbers); // Prints [2.5, 5, 8, 10]
```

Note: The sorting algorithm used is not stable [https://en.wikipedia.org/wiki/Sorting_algorithm#Stability]. This means that equivalent elements (such as `2` and `2.0`) may have their order changed when calling sort.

---

void **sort_custom**(func: Callable)

Sorts the array using a custom Callable.

`func` is called as many times as necessary, receiving two array elements as arguments. The function should return `true` if the first element should be moved *before* the second one, otherwise it should return `false`.

```
func sort_ascending(a, b):
    if a[1] < b[1]:
        return true
    return false

func _ready():
    var my_items = [["Tomato", 5], ["Apple", 9], ["Rice", 4]]
    my_items.sort_custom(sort_ascending)
    print(my_items) # Prints [["Rice", 4], ["Tomato", 5], ["Apple", 9]]

    # Sort descending, using a lambda function.
    my_items.sort_custom(func(a, b): return a[1] > b[1])
    print(my_items) # Prints [["Apple", 9], ["Tomato", 5], ["Rice", 4]]
```

It may also be necessary to use this method to sort strings by natural order, with String.naturalnocasecmp_to, as in the following example:

```
var files = ["newfile1", "newfile2", "newfile10", "newfile11"]
files.sort_custom(func(a, b): return a.naturalnocasecmp_to(b) < 0)
print(files) # Prints ["newfile1", "newfile2", "newfile10", "newfile11"]
```

Note: In C#, this method is not supported.

Note: The sorting algorithm used is not stable [https://en.wikipedia.org/wiki/Sorting_algorithm#Stability]. This means that values considered equal may have their order changed when calling this method.

Note: You should not randomize the return value of `func`, as the heapsort algorithm expects a consistent result. Randomizing the return value will result in unexpected behavior.

Operator Descriptions

bool **operator !=**(right: Array) 🔗

Returns `true` if the array's size or its elements are different than `right`'s.

Array **operator +**(right: Array) 🔗

Appends the `right` array to the left operand, creating a new **Array**. This is also known as an array concatenation.

GDScript

```
var array1 = ["One", 2]
var array2 = [3, "Four"]
print(array1 + array2) # Prints ["One", 2, 3, "Four"]
```

C#

```
// Note that concatenation is not possible with C#'s native
Array type.
var array1 = new Godot.Collections.Array{"One", 2};
var array2 = new Godot.Collections.Array{3, "Four"};
GD.Print(array1 + array2); // Prints ["One", 2, 3, "Four"]
```

Note: For existing arrays, append_array is much more efficient than concatenation and assignment with the `+=` operator.

---

bool **operator <**(right: Array) 🔗

Compares the elements of both arrays in order, starting from index `0` and ending on the last index in common between both arrays. For each pair of elements, returns `true` if this array's element is less than `right`'s, `false` if this element is greater. Otherwise, continues to the next pair.

If all searched elements are equal, returns `true` if this array's size is less than `right`'s, otherwise returns `false`.

---

bool **operator <=**(right: Array)

Compares the elements of both arrays in order, starting from index `0` and ending on the last index in common between both arrays. For each pair of elements, returns `true` if this array's element is less than `right`'s, `false` if this element is greater. Otherwise, continues to the next pair.

If all searched elements are equal, returns `true` if this array's size is less or equal to `right`'s, otherwise returns `false`.

---

bool **operator ==**(right: Array)

Compares the left operand **Array** against the `right` **Array**. Returns `true` if the sizes and contents of the arrays are equal, `false` otherwise.

---

bool **operator >**(right: Array)

Compares the elements of both arrays in order, starting from index `0` and ending on the last index in common between both arrays. For each pair of elements, returns `true` if this array's element is greater than `right`'s, `false` if this element is less. Otherwise, continues to the next pair.

If all searched elements are equal, returns `true` if this array's size is greater than `right`'s, otherwise returns `false`.

---

bool **operator >=**(right: Array)

Compares the elements of both arrays in order, starting from index `0` and ending on the last index in common between both arrays. For each pair of elements, returns `true` if this array's element is greater than `right`'s, `false` if this element is less. Otherwise, continues to the next pair.

If all searched elements are equal, returns `true` if this array's size is greater or equal to `right`'s, otherwise returns `false`.

Variant **operator []**(index: int)

Returns the Variant element at the specified `index`. Arrays start at index 0. If `index` is greater or equal to `0`, the element is fetched starting from the beginning of the array. If `index` is a negative value, the element is fetched starting from the end. Accessing an array out-of-bounds will cause a run-time error, pausing the project execution if run from the editor.

Basis

A 3×3 matrix for representing 3D rotation and scale.

Description

The **Basis** built-in Variant type is a 3×3 matrix [https://en.wikipedia.org/wiki/Matrix_(mathematics)] used to represent 3D rotation, scale, and shear. It is frequently used within a Transform3D.

A **Basis** is composed by 3 axis vectors, each representing a column of the matrix: x, y, and z. The length of each axis (Vector3.length) influences the basis's scale, while the direction of all axes influence the rotation. Usually, these axes are perpendicular to one another. However, when you rotate any axis individually, the basis becomes sheared. Applying a sheared basis to a 3D model will make the model appear distorted.

A **Basis** is **orthogonal** if its axes are perpendicular to each other. A basis is **normalized** if the length of every axis is `1`. A basis is **uniform** if all axes share the same length (see get_scale). A basis is **orthonormal** if it is both orthogonal and normalized, which allows it to only represent rotations. A basis is **conformal** if it is both orthogonal and uniform, which ensures it is not distorted.

For a general introduction, see the Matrices and transforms tutorial.

Note: Godot uses a right-handed coordinate system [https://en.wikipedia.org/wiki/Right-hand_rule], which is a common standard. For directions, the convention for built-in types like Camera3D is for -Z to point forward (+X is right, +Y is up, and +Z is back). Other objects may use different direction conventions. For more information, see the 3D asset direction conventions tutorial.

Note: The basis matrices are exposed as column-major [https://www.mindcontrol.org/~hplus/graphics/matrix-layout.html] order, which is the same as OpenGL. However, they are stored internally in row-major order, which is the same as DirectX.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Math documentation index
- Matrices and transforms
- Using 3D transforms
- Matrix Transform Demo [https://godotengine.org/asset-library/asset/2787]
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]
- 2.5D Game Demo [https://godotengine.org/asset-library/asset/2783]

Properties

Vector3	x	`Vector3(1, 0, 0)`
Vector3	y	`Vector3(0, 1, 0)`

Vector3	z	`Vector3(0, 0, 1)`

Constructors

Basis	Basis()
Basis	Basis(from: Basis)
Basis	Basis(axis: Vector3, angle: float)
Basis	Basis(from: Quaternion)
Basis	Basis(x_axis: Vector3, y_axis: Vector3, z_axis: Vector3)

Methods

float	determinant() const
Basis	from_euler(euler: Vector3, order: int = 2) static
Basis	from_scale(scale: Vector3) static
Vector3	get_euler(order: int = 2) const

Quaternion	get_rotation_quaternion() const
Vector3	get_scale() const
Basis	inverse() const
bool	is_conformal() const
bool	is_equal_approx(b: Basis) const
bool	is_finite() const
Basis	looking_at(target: Vector3, up: Vector3 = Vector3(0, 1, 0), use_model_front: bool = false) static
Basis	orthonormalized() const
Basis	rotated(axis: Vector3, angle: float) const
Basis	scaled(scale: Vector3) const
Basis	slerp(to: Basis, weight: float) const
float	tdotx(with: Vector3) const

float	tdoty(with: Vector3) const
float	tdotz(with: Vector3) const
Basis	transposed() const

Operators

bool	operator !=(right: Basis)
Basis	operator *(right: Basis)
Vector3	operator *(right: Vector3)
Basis	operator *(right: float)
Basis	operator *(right: int)
Basis	operator /(right: float)
Basis	operator /(right: int)
bool	operator ==(right: Basis)

Vector3 operator [](index: int)

Constants

IDENTITY = `Basis(1, 0, 0, 0, 1, 0, 0, 0, 1)` 🔗

The identity basis. This is a basis with no rotation, no shear, and its scale being `1`. This means that:

- The x points right (Vector3.RIGHT);
- The y points up (Vector3.UP);
- The z points back (Vector3.BACK).

```
var basis := Basis.IDENTITY
print("| X | Y | Z")
print("| %s | %s | %s" % [basis.x.x, basis.y.x, basis.z.x])
print("| %s | %s | %s" % [basis.x.y, basis.y.y, basis.z.y])
print("| %s | %s | %s" % [basis.x.z, basis.y.z, basis.z.z])
# Prints:
# | X | Y | Z
# | 1 | 0 | 0
# | 0 | 1 | 0
# | 0 | 0 | 1
```

This is identical to creating Basis without any parameters. This constant can be used to make your code clearer, and for consistency with C#.

FLIP_X = `Basis(-1, 0, 0, 0, 1, 0, 0, 0, 1)` 🔗

When any basis is multiplied by FLIP_X, it negates all components of the x axis (the X column).

When FLIP_X is multiplied by any basis, it negates the Vector3.x component of all axes (the X row).

FLIP_Y = `Basis(1, 0, 0, 0, -1, 0, 0, 0, 1)`

When any basis is multiplied by FLIP_Y, it negates all components of the y axis (the Y column).

When FLIP_Y is multiplied by any basis, it negates the Vector3.y component of all axes (the Y row).

FLIP_Z = `Basis(1, 0, 0, 0, 1, 0, 0, 0, -1)`

When any basis is multiplied by FLIP_Z, it negates all components of the z axis (the Z column).

When FLIP_Z is multiplied by any basis, it negates the Vector3.z component of all axes (the Z row).

Property Descriptions

Vector3 **x** = `Vector3(1, 0, 0)`

The basis's X axis, and the column `0` of the matrix.

On the identity basis, this vector points right (Vector3.RIGHT).

Vector3 **y** = `Vector3(0, 1, 0)`

The basis's Y axis, and the column `1` of the matrix.

On the identity basis, this vector points up (Vector3.UP).

Vector3 **z** = `Vector3(0, 0, 1)`

The basis's Z axis, and the column `2` of the matrix.

On the identity basis, this vector points back (Vector3.BACK).

Constructor Descriptions

Basis **Basis**()

Constructs a **Basis** identical to the IDENTITY.

Basis **Basis**(from: Basis)

Constructs a **Basis** as a copy of the given **Basis**.

Basis **Basis**(axis: Vector3, angle: float)

Constructs a **Basis** that only represents rotation, rotated around the `axis` by the given `angle`, in radians. The axis must be a normalized vector.

Note: This is the same as using rotated on the IDENTITY basis. With more than one angle consider using from_euler, instead.

Basis **Basis**(from: Quaternion)

Constructs a **Basis** that only represents rotation from the given Quaternion.

Note: Quaternions *only* store rotation, not scale. Because of this, conversions from **Basis** to Quaternion cannot always be reversed.

Basis **Basis**(x_axis: Vector3, y_axis: Vector3, z_axis: Vector3)

Constructs a **Basis** from 3 axis vectors. These are the columns of the basis matrix.

Method Descriptions

float **determinant**() const

Returns the determinant [https://en.wikipedia.org/wiki/Determinant] of this basis's matrix. For advanced math, this number can be used to determine a few attributes:

- If the determinant is exactly `0`, the basis is not invertible (see inverse).
- If the determinant is a negative number, the basis represents a negative scale.

Note: If the basis's scale is the same for every axis, its determinant is always that scale by the power of 2.

Basis **from_euler**(euler: Vector3, order: int = 2) static

Constructs a new **Basis** that only represents rotation from the given Vector3 of Euler angles [https://en.wikipedia.org/wiki/Euler_angles], in radians.

- The Vector3.x should contain the angle around the x axis (pitch).
- The Vector3.y should contain the angle around the y axis (yaw).
- The Vector3.z should contain the angle around the z axis (roll).

GDScript

```
# Creates a Basis whose z axis points down.
var my_basis = Basis.from_euler(Vector3(TAU / 4, 0, 0))
```

```
print(my_basis.z) # Prints (0, -1, 0).
```

C#

```
// Creates a Basis whose z axis points down.
var myBasis = Basis.FromEuler(new Vector3(Mathf.Tau / 4.0f,
0.0f, 0.0f));

GD.Print(myBasis.Z); // Prints (0, -1, 0).
```

The order of each consecutive rotation can be changed with `order` (see EulerOrder constants). By default, the YXZ convention is used (@GlobalScope.EULER_ORDER_YXZ): the basis rotates first around the Y axis (yaw), then X (pitch), and lastly Z (roll). When using the opposite method get_euler, this order is reversed.

Basis **from_scale**(scale: Vector3) static

Constructs a new **Basis** that only represents scale, with no rotation or shear, from the given `scale` vector.

GDScript

```
var my_basis = Basis.from_scale(Vector3(2, 4, 8))

print(my_basis.x) # Prints (2, 0, 0).
print(my_basis.y) # Prints (0, 4, 0).
print(my_basis.z) # Prints (0, 0, 8).
```

C#

```
var myBasis = Basis.FromScale(new Vector3(2.0f, 4.0f, 8.0f));

GD.Print(myBasis.X); // Prints (2, 0, 0).
GD.Print(myBasis.Y); // Prints (0, 4, 0).
GD.Print(myBasis.Z); // Prints (0, 0, 8).
```

Note: In linear algebra, the matrix of this basis is also known as a diagonal matrix [https://en.wikipedia.org/wiki/Diagonal_matrix].

Vector3 **get_euler**(order: int = 2) const

Returns this basis's rotation as a Vector3 of Euler angles [https://en.wikipedia.org/wiki/Euler_angles], in radians.

- The Vector3.x contains the angle around the x axis (pitch);
- The Vector3.y contains the angle around the y axis (yaw);
- The Vector3.z contains the angle around the z axis (roll).

The order of each consecutive rotation can be changed with `order` (see EulerOrder constants). By default, the YXZ convention is used (@GlobalScope.EULER_ORDER_YXZ): Z (roll) is calculated first, then X (pitch), and lastly Y (yaw). When using the opposite method from_euler, this order is reversed.

Note: Euler angles are much more intuitive but are not suitable for 3D math. Because of this, consider using the get_rotation_quaternion method instead, which returns a Quaternion.

Note: In the Inspector dock, a basis's rotation is often displayed in Euler angles (in degrees), as is the case with the Node3D.rotation property.

Quaternion **get_rotation_quaternion**() const

Returns this basis's rotation as a Quaternion.

Note: Quatenions are much more suitable for 3D math but are less intuitive. For user interfaces, consider using the get_euler method, which returns Euler angles.

Vector3 **get_scale**() const 🔗

Returns the length of each axis of this basis, as a Vector3. If the basis is not sheared, this is the scaling factor. It is not affected by rotation.

GDScript

```
var my_basis = Basis(
    Vector3(2, 0, 0),
    Vector3(0, 4, 0),
    Vector3(0, 0, 8)
)
# Rotating the Basis in any way preserves its scale.
my_basis = my_basis.rotated(Vector3.UP, TAU / 2)
my_basis = my_basis.rotated(Vector3.RIGHT, TAU / 4)

print(my_basis.get_scale()) # Prints (2, 4, 8).
```

C#

```
var myBasis = new Basis(
    Vector3(2.0f, 0.0f, 0.0f),
    Vector3(0.0f, 4.0f, 0.0f),
    Vector3(0.0f, 0.0f, 8.0f)
);
// Rotating the Basis in any way preserves its scale.
myBasis = myBasis.Rotated(Vector3.Up, Mathf.Tau / 2.0f);
myBasis = myBasis.Rotated(Vector3.Right, Mathf.Tau / 4.0f);

GD.Print(myBasis.Scale); // Prints (2, 4, 8).
```

Note: If the value returned by determinant is negative, the scale is also negative.

Basis **inverse**() const 🔗

Returns the inverse of this basis's matrix [https://en.wikipedia.org/wiki/Invertible_matrix].

---

bool **is_conformal**() const

Returns `true` if this basis is conformal. A conformal basis is both *orthogonal* (the axes are perpendicular to each other) and *uniform* (the axes share the same length). This method can be especially useful during physics calculations.

---

bool **is_equal_approx**(b: Basis) const

Returns `true` if this basis and `b` are approximately equal, by calling @GlobalScope.is_equal_approx on all vector components.

---

bool **is_finite**() const

Returns `true` if this basis is finite, by calling @GlobalScope.is_finite on all vector components.

Basis **looking_at**(target: Vector3, up: Vector3 = Vector3(0, 1, 0), use_model_front: bool = false) static

Creates a new **Basis** with a rotation such that the forward axis (-Z) points towards the `target` position.

By default, the -Z axis (camera forward) is treated as forward (implies +X is right). If `use_model_front` is `true`, the +Z axis (asset front) is treated as forward (implies +X is left) and points toward the `target` position.

The up axis (+Y) points as close to the `up` vector as possible while staying perpendicular to the forward axis. The returned basis is orthonormalized (see orthonormalized). The `target` and `up` vectors cannot be Vector3.ZERO, and cannot be parallel to each other.

Basis **orthonormalized**() const

Returns the orthonormalized version of this basis. An orthonormal basis is both *orthogonal* (the axes are perpendicular to each other) and *normalized* (the axes have a length of `1`), which also means it can only represent rotation.

It is often useful to call this method to avoid rounding errors on a rotating basis:

GDScript

```
# Rotate this Node3D every frame.
func _process(delta):
    basis = basis.rotated(Vector3.UP, TAU * delta)
    basis = basis.rotated(Vector3.RIGHT, TAU * delta)

    basis = basis.orthonormalized()
```

C#

```
// Rotate this Node3D every frame.
public override void _Process(double delta)
{
    Basis = Basis.Rotated(Vector3.Up, Mathf.Tau * (float)delta)
                 .Rotated(Vector3.Right, Mathf.Tau *
(float)delta)
                 .Orthonormalized();
}
```

Basis **rotated**(axis: Vector3, angle: float) const

Returns this basis rotated around the given `axis` by `angle` (in radians). The `axis` must be a normalized vector (see Vector3.normalized).

Positive values rotate this basis clockwise around the axis, while negative values rotate it counterclockwise.

GDScript

```
var my_basis = Basis.IDENTITY
var angle = TAU / 2

my_basis = my_basis.rotated(Vector3.UP, angle)    # Rotate
around the up axis (yaw).
my_basis = my_basis.rotated(Vector3.RIGHT, angle) # Rotate
around the right axis (pitch).
my_basis = my_basis.rotated(Vector3.BACK, angle)  # Rotate
around the back axis (roll).
```

C#

```
var myBasis = Basis.Identity;
var angle = Mathf.Tau / 2.0f;

myBasis = myBasis.Rotated(Vector3.Up, angle);    // Rotate
around the up axis (yaw).
myBasis = myBasis.Rotated(Vector3.Right, angle); // Rotate
around the right axis (pitch).
myBasis = myBasis.Rotated(Vector3.Back, angle);  // Rotate
around the back axis (roll).
```

Basis **scaled**(scale: Vector3) const

Returns this basis with each axis's components scaled by the given `scale`'s components.

The basis matrix's rows are multiplied by `scale`'s components. This operation is a global scale (relative to the parent).

GDScript

```
var my_basis = Basis(
    Vector3(1, 1, 1),
    Vector3(2, 2, 2),
    Vector3(3, 3, 3)
)
my_basis = my_basis.scaled(Vector3(0, 2, -2))

print(my_basis.x) # Prints (0, 2, -2).
print(my_basis.y) # Prints (0, 4, -4).
print(my_basis.z) # Prints (0, 6, -6).
```

C#

```
var myBasis = new Basis(
    new Vector3(1.0f, 1.0f, 1.0f),
    new Vector3(2.0f, 2.0f, 2.0f),
    new Vector3(3.0f, 3.0f, 3.0f)
);
myBasis = myBasis.Scaled(new Vector3(0.0f, 2.0f, -2.0f));

GD.Print(myBasis.X); // Prints (0, 2, -2).
GD.Print(myBasis.Y); // Prints (0, 4, -4).
GD.Print(myBasis.Z); // Prints (0, 6, -6).
```

Basis **slerp**(to: Basis, weight: float) const

Performs a spherical-linear interpolation with the `to` basis, given a `weight`. Both this basis and `to` should represent a rotation.

Example: Smoothly rotate a Node3D to the target basis over time, with a Tween.

```
var start_basis = Basis.IDENTITY
var target_basis = Basis.IDENTITY.rotated(Vector3.UP, TAU / 2)

func _ready():
    create_tween().tween_method(interpolate, 0.0, 1.0,
5.0).set_trans(Tween.TRANS_EXPO)
```

```
func interpolate(weight):
    basis = start_basis.slerp(target_basis, weight)
```

---

float **tdotx**(with: Vector3) const

Returns the transposed dot product between `with` and the x axis (see transposed).

This is equivalent to `basis.x.dot(vector)`.

---

float **tdoty**(with: Vector3) const

Returns the transposed dot product between `with` and the y axis (see transposed).

This is equivalent to `basis.y.dot(vector)`.

---

float **tdotz**(with: Vector3) const

Returns the transposed dot product between `with` and the z axis (see transposed).

This is equivalent to `basis.z.dot(vector)`.

Basis **transposed**() const

Returns the transposed version of this basis. This turns the basis matrix's columns into rows, and its rows into columns.

GDScript

```
var my_basis = Basis(
    Vector3(1, 2, 3),
    Vector3(4, 5, 6),
    Vector3(7, 8, 9)
)
my_basis = my_basis.transposed()

print(my_basis.x) # Prints (1, 4, 7).
print(my_basis.y) # Prints (2, 5, 8).
print(my_basis.z) # Prints (3, 6, 9).
```

C#

```
var myBasis = new Basis(
    new Vector3(1.0f, 2.0f, 3.0f),
    new Vector3(4.0f, 5.0f, 6.0f),
    new Vector3(7.0f, 8.0f, 9.0f)
);
myBasis = myBasis.Transposed();

GD.Print(myBasis.X); // Prints (1, 4, 7).
GD.Print(myBasis.Y); // Prints (2, 5, 8).
GD.Print(myBasis.Z); // Prints (3, 6, 9).
```

Operator Descriptions

bool **operator !=**(right: Basis) 🔗

Returns `true` if the components of both **Basis** matrices are not equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Basis **operator ***(right: Basis) 🔗

Transforms (multiplies) the `right` basis by this basis.

This is the operation performed between parent and child Node3Ds.

Vector3 **operator** *(right: Vector3)

Transforms (multiplies) the `right` vector by this basis, returning a Vector3.

GDScript

```
# Basis that swaps the X/Z axes and doubles the scale.
var my_basis = Basis(Vector3(0, 2, 0), Vector3(2, 0, 0),
Vector3(0, 0, 2))
print(my_basis * Vector3(1, 2, 3)) # Prints (4, 2, 6)
```

C#

```
// Basis that swaps the X/Z axes and doubles the scale.
var myBasis = new Basis(new Vector3(0, 2, 0), new Vector3(2, 0,
0), new Vector3(0, 0, 2));
GD.Print(myBasis * new Vector3(1, 2, 3)); // Prints (4, 2, 6)
```

Basis **operator** *(right: float)

Multiplies all components of the **Basis** by the given float. This affects the basis's scale uniformly, resizing all 3 axes by the `right` value.

Basis **operator** *(right: int)

Multiplies all components of the **Basis** by the given int. This affects the basis's scale uniformly, resizing all 3 axes by the `right` value.

Basis **operator** /(right: float)

Divides all components of the **Basis** by the given float. This affects the basis's scale uniformly, resizing all 3 axes by the `right` value.

Basis **operator /**(right: int) 🔗

Divides all components of the **Basis** by the given int. This affects the basis's scale uniformly, resizing all 3 axes by the `right` value.

---

bool **operator ==**(right: Basis) 🔗

Returns `true` if the components of both **Basis** matrices are exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Vector3 **operator []**(index: int) 🔗

Accesses each axis (column) of this basis by their index. Index `0` is the same as x, index `1` is the same as y, and index `2` is the same as z.

Note: In C++, this operator accesses the rows of the basis matrix, *not* the columns. For the same behavior as scripting languages, use the `set_column` and `get_column` methods.

bool

A built-in boolean type.

Description

The **bool** is a built-in Variant type that may only store one of two values: `true` or `false`. You can imagine it as a switch that can be either turned on or off, or as a binary digit that can either be 1 or 0.

Booleans can be directly used in `if`, and other conditional statements:

GDScript

```
var can_shoot = true
if can_shoot:
    launch_bullet()
```

C#

```
bool canShoot = true;
if (canShoot)
{
    LaunchBullet();
}
```

All comparison operators return booleans (`==`, `>`, `<=`, etc.). As such, it is not necessary to compare booleans themselves. You do not need to add `== true` or `== false`.

Booleans can be combined with the logical operators `and`, `or`, `not` to create complex conditions:

GDScript

```
if bullets > 0 and not is_reloading():
    launch_bullet()

if bullets == 0 or is_reloading():
    play_clack_sound()
```

C#

```
if (bullets > 0 && !IsReloading())
{
    LaunchBullet();
}

if (bullets == 0 || IsReloading())
{
    PlayClackSound();
}
```

Note: In modern programming languages, logical operators are evaluated in order. All remaining conditions are skipped if their result would have no effect on the final value. This concept is known as short-circuit evaluation [https://en.wikipedia.org/wiki/Short-circuit_evaluation] and can be useful to avoid evaluating expensive conditions in some performance-critical cases.

Note: By convention, built-in methods and properties that return booleans are usually defined as yes-no questions, single adjectives, or similar (String.is_empty, Node.can_process, Camera2D.enabled, etc.).

Constructors

bool bool()

---

bool bool(from: bool)

---

bool	bool(from: float)
bool	bool(from: int)

Operators

bool	operator !=(right: bool)
bool	operator <(right: bool)
bool	operator ==(right: bool)
bool	operator >(right: bool)

Constructor Descriptions

bool **bool**() 🔗

Constructs a **bool** set to `false`.

---

bool **bool**(from: bool)

Constructs a **bool** as a copy of the given **bool**.

---

bool **bool**(from: float)

Cast a float value to a boolean value. Returns `false` if `from` is equal to `0.0` (including `-0.0`), and `true` for all other values (including @GDScript.INF and @GDScript.NAN).

---

bool **bool**(from: int)

Cast an int value to a boolean value. Returns `false` if `from` is equal to `0`, and `true` for all other values.

Operator Descriptions

bool **operator !=**(right: bool) 🔗

Returns `true` if the two booleans are not equal. That is, one is `true` and the other is `false`. This operation can be seen as a logical XOR.

---

bool **operator <**(right: bool) 🔗

Returns `true` if the left operand is `false` and the right operand is `true`.

---

bool **operator ==**(right: bool) 🔗

Returns `true` if the two booleans are equal. That is, both are `true` or both are `false`. This operation can be seen as a logical EQ or XNOR.

---

bool **operator >**(right: bool) 🔗

Returns `true` if the left operand is `true` and the right operand is `false`.

Callable

A built-in type representing a method or a standalone function.

Description

Callable is a built-in Variant type that represents a function. It can either be a method within an Object instance, or a custom callable used for different purposes (see is_custom). Like all Variant types, it can be stored in variables and passed to other functions. It is most commonly used for signal callbacks.

Example:

GDScript

```
func print_args(arg1, arg2, arg3 = ""):
    prints(arg1, arg2, arg3)

func test():
    var callable = Callable(self, "print_args")
    callable.call("hello", "world")  # Prints "hello world ".
    callable.call(Vector2.UP, 42, callable)  # Prints "(0, -1) 42 Node(node.gd)::print_args".
    callable.call("invalid")  # Invalid call, should have at least 2 arguments.
```

C#

```
// Default parameter values are not supported.
public void PrintArgs(Variant arg1, Variant arg2, Variant arg3 = default)
{
    GD.PrintS(arg1, arg2, arg3);
}
```

```
public void Test()
{
    // Invalid calls fail silently.
    Callable callable = new Callable(this, MethodName.PrintArgs);
    callable.Call("hello", "world"); // Default parameter values are not supported, should have 3 arguments.
    callable.Call(Vector2.Up, 42, callable); // Prints "(0, -1) 42 Node(Node.cs)::PrintArgs".
    callable.Call("invalid"); // Invalid call, should have 3 arguments.
}
```

In GDScript, it's possible to create lambda functions within a method. Lambda functions are custom callables that are not associated with an Object instance. Optionally, lambda functions can also be named. The name will be displayed in the debugger, or when calling get_method.

```
func _init():
    var my_lambda = func (message):
        print(message)

    # Prints Hello everyone!
    my_lambda.call("Hello everyone!")

    # Prints "Attack!", when the button_pressed signal is emitted.
    button_pressed.connect(func(): print("Attack!"))
```

In GDScript, you can access methods and global functions as **Callable**s:

```
tween.tween_callback(node.queue_free)  # Object methods.
tween.tween_callback(array.clear)  # Methods of built-in types.
tween.tween_callback(print.bind("Test"))  # Global functions.
```

Note: Dictionary does not support the above due to ambiguity with keys.

```
var dictionary = {"hello": "world"}

# This will not work, `clear` is treated as a key.
tween.tween_callback(dictionary.clear)

# This will work.
tween.tween_callback(Callable.create(dictionary, "clear"))
```

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

Callable	Callable()
Callable	Callable(from: Callable)
Callable	Callable(object: Object, method: StringName)

Methods

Callable	bind(...) vararg const
Callable	bindv(arguments: Array)
Variant	call(...) vararg const

void	call_deferred(...) vararg const
Variant	callv(arguments: Array) const
Callable	create(variant: Variant, method: StringName) static
int	get_argument_count() const
Array	get_bound_arguments() const
int	get_bound_arguments_count() const
StringName	get_method() const
Object	get_object() const
int	get_object_id() const
int	hash() const
bool	is_custom() const
bool	is_null() const
bool	is_standard() const

bool	is_valid() const
void	rpc(...) vararg const
void	rpc_id(peer_id: int, ...) vararg const
Callable	unbind(argcount: int) const

Operators

bool	operator !=(right: Callable)
bool	operator ==(right: Callable)

Constructor Descriptions

Callable **Callable**() 🔗

Constructs an empty **Callable**, with no object nor method bound.

Callable **Callable**(from: Callable)

Constructs a **Callable** as a copy of the given **Callable**.

Callable **Callable**(object: Object, method: StringName)

Creates a new **Callable** for the method named `method` in the specified `object`.

Note: For methods of built-in Variant types, use create instead.

Method Descriptions

Callable **bind**(...) vararg const 🔗

Returns a copy of this **Callable** with one or more arguments bound. When called, the bound arguments are passed *after* the arguments supplied by call. See also unbind.

Note: When this method is chained with other similar methods, the order in which the argument list is modified is read from right to left.

Callable **bindv**(arguments: Array) 🔗

Returns a copy of this **Callable** with one or more arguments bound, reading them from an array. When called, the bound arguments are passed *after* the arguments supplied by call. See also unbind.

Note: When this method is chained with other similar methods, the order in which the argument list is modified is read from right to left.

Variant **call**(...) vararg const 🔗

Calls the method represented by this **Callable**. Arguments can be passed and should match the method's signature.

---

void **call_deferred**(...) vararg const 🔗

Calls the method represented by this **Callable** in deferred mode, i.e. at the end of the current frame. Arguments can be passed and should match the method's signature.

GDScript

```
func _ready():
    grab_focus.call_deferred()
```

C#

```
public override void _Ready()
{
    Callable.From(GrabFocus).CallDeferred();
}
```

Note: Deferred calls are processed at idle time. Idle time happens mainly at the end of process and physics frames. In it, deferred calls will be run until there are none left, which means you can defer calls from other deferred calls and they'll still be run in the current idle time cycle. This means you should not call a method deferred from itself (or from a method called by it), as this causes infinite recursion the same way as if you had called the method directly.

See also Object.call_deferred.

Variant **callv**(arguments: Array) const 🔗

Calls the method represented by this **Callable**. Unlike call, this method expects all arguments to be contained inside the `arguments` Array.

Callable **create**(variant: Variant, method: StringName) static 🔗

Creates a new **Callable** for the method named `method` in the specified `variant`. To represent a method of a built-in Variant type, a custom callable is used (see is_custom). If `variant` is Object, then a standard callable will be created instead.

Note: This method is always necessary for the Dictionary type, as property syntax is used to access its entries. You may also use this method when `variant`'s type is not known in advance (for polymorphism).

int **get_argument_count**() const 🔗

Returns the total number of arguments this **Callable** should take, including optional arguments. This means that any arguments bound with bind are *subtracted* from the result, and any arguments unbound with unbind are *added* to the result.

Array **get_bound_arguments**() const 🔗

Return the bound arguments (as long as get_bound_arguments_count is greater than zero), or empty (if get_bound_arguments_count is less than or equal to zero).

int **get_bound_arguments_count**() const 🔗

Returns the total amount of arguments bound (or unbound) via successive bind or unbind calls. If the amount of arguments unbound is greater than the ones bound, this function returns a value less than zero.

StringName **get_method**() const 🔗

Returns the name of the method represented by this **Callable**. If the callable is a GDScript lambda function, returns the function's name or `"<anonymous lambda>"`.

Object **get_object**() const 🔗

Returns the object on which this **Callable** is called.

---

int **get_object_id**() const 🔗

Returns the ID of this **Callable**'s object (see Object.get_instance_id).

---

int **hash**() const 🔗

Returns the 32-bit hash value of this **Callable**'s object.

Note: Callables with equal content will always produce identical hash values. However, the reverse is not true. Returning identical hash values does *not* imply the callables are equal, because different callables can have identical hash values due to hash collisions. The engine uses a 32-bit hash algorithm for hash.

---

bool **is_custom**() const 🔗

Returns `true` if this **Callable** is a custom callable. Custom callables are used:

- for binding/unbinding arguments (see bind and unbind);
- for representing methods of built-in Variant types (see create);
- for representing global, lambda, and RPC functions in GDScript;
- for other purposes in the core, GDExtension, and C#.

---

bool **is_null**() const

Returns `true` if this **Callable** has no target to call the method on.

---

bool **is_standard**() const

Returns `true` if this **Callable** is a standard callable. This method is the opposite of is_custom. Returns `false` if this callable is a lambda function.

---

bool **is_valid**() const

Returns `true` if the callable's object exists and has a valid method name assigned, or is a custom callable.

---

void **rpc**(...) vararg const

Perform an RPC (Remote Procedure Call) on all connected peers. This is used for multiplayer and is normally not available, unless the function being called has been marked as *RPC* (using @GDScript.@rpc or Node.rpc_config). Calling this method on unsupported functions will result in an error. See Node.rpc.

---

void **rpc_id**(peer_id: int, ...) vararg const

Perform an RPC (Remote Procedure Call) on a specific peer ID (see multiplayer documentation for reference). This is used for multiplayer and is normally not available unless the function being called has been marked as *RPC* (using @GDScript.@rpc or Node.rpc_config). Calling this method on unsupported functions will result in an error. See Node.rpc_id.

Callable **unbind**(argcount: int) const

Returns a copy of this **Callable** with a number of arguments unbound. In other words, when the new callable is called the last few arguments supplied by the user are ignored, according to `argcount`. The remaining arguments are passed to the callable. This allows to use the original callable in a context that attempts to pass more arguments than this callable can handle, e.g. a signal with a fixed number of arguments. See also bind.

Note: When this method is chained with other similar methods, the order in which the argument list is modified is read from right to left.

```
func _ready():
    foo.unbind(1).call(1, 2) # Calls foo(1).
    foo.bind(3, 4).unbind(1).call(1, 2) # Calls foo(1, 3, 4), note that it does not change the arguments from bind.
```

Operator Descriptions

bool **operator !=**(right: Callable)

Returns `true` if both **Callable**s invoke different targets.

bool **operator ==**(right: Callable)

Returns `true` if both **Callable**s invoke the same custom target.

Color

A color represented in RGBA format.

Description

A color represented in RGBA format by a red (r), green (g), blue (b), and alpha (a) component. Each component is a 32-bit floating-point value, usually ranging from `0.0` to `1.0`. Some properties (such as CanvasItem.modulate) may support values greater than `1.0`, for overbright or HDR (High Dynamic Range) colors.

Colors can be created in various ways: By the various **Color** constructors, by static methods such as from_hsv, and by using a name from the set of standardized colors based on X11 color names [https://en.wikipedia.org/wiki/X11_color_names] with the addition of TRANSPARENT. GDScript also provides @GDScript.Color8, which uses integers from `0` to `255` and doesn't support overbright colors.

Note: In a boolean context, a Color will evaluate to `false` if it is equal to `Color(0, 0, 0, 1)` (opaque black). Otherwise, a Color will always evaluate to `true`.

Color constants cheatsheet [https://raw.githubusercontent.com/godotengine/godot-docs/master/img/color_constants.png]

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- 2D GD Paint Demo [https://godotengine.org/asset-library/asset/2768]
- Tween Interpolation Demo [https://godotengine.org/asset-library/asset/2733]
- GUI Drag And Drop Demo [https://godotengine.org/asset-library/asset/2767]

Properties

float	a	`1.0`
int	a8	`255`
float	b	`0.0`
int	b8	`0`
float	g	`0.0`
int	g8	`0`
float	h	`0.0`
float	r	`0.0`

int	r8	`0`
float	s	`0.0`
float	v	`0.0`

Constructors

Color	Color()
Color	Color(from: Color, alpha: float)
Color	Color(from: Color)
Color	Color(code: String)
Color	Color(code: String, alpha: float)
Color	Color(r: float, g: float, b: float)
Color	Color(r: float, g: float, b: float, a: float)

Methods

Color	blend(over: Color) const
Color	clamp(min: Color = Color(0, 0, 0, 0), max: Color = Color(1, 1, 1, 1)) const
Color	darkened(amount: float) const
Color	from_hsv(h: float, s: float, v: float, alpha: float = 1.0) static
Color	from_ok_hsl(h: float, s: float, l: float, alpha: float = 1.0) static
Color	from_rgbe9995(rgbe: int) static
Color	from_string(str: String, default: Color) static
float	get_luminance() const
Color	hex(hex: int) static
Color	hex64(hex: int) static
Color	html(rgba: String) static
bool	html_is_valid(color: String) static

Color	inverted() const
bool	is_equal_approx(to: Color) const
Color	lerp(to: Color, weight: float) const
Color	lightened(amount: float) const
Color	linear_to_srgb() const
Color	srgb_to_linear() const
int	to_abgr32() const
int	to_abgr64() const
int	to_argb32() const
int	to_argb64() const
String	to_html(with_alpha: bool = true) const
int	to_rgba32() const
int	to_rgba64() const

Operators

bool	operator !=(right: Color)
Color	operator *(right: Color)
Color	operator *(right: float)
Color	operator *(right: int)
Color	operator +(right: Color)
Color	operator -(right: Color)
Color	operator /(right: Color)
Color	operator /(right: float)
Color	operator /(right: int)
bool	operator ==(right: Color)
float	operator [](index: int)

Color operator unary+()

Color operator unary-()

Constants

ALICE_BLUE = `Color(0.941176, 0.972549, 1, 1)`

Alice blue color.

ANTIQUE_WHITE = `Color(0.980392, 0.921569, 0.843137, 1)`

Antique white color.

AQUA = `Color(0, 1, 1, 1)`

Aqua color.

AQUAMARINE = `Color(0.498039, 1, 0.831373, 1)`

Aquamarine color.

AZURE = `Color(0.941176, 1, 1, 1)`

Azure color.

BEIGE = `Color(0.960784, 0.960784, 0.862745, 1)`

Beige color.

BISQUE = `Color(1, 0.894118, 0.768627, 1)`

Bisque color.

BLACK = `Color(0, 0, 0, 1)`

Black color. In GDScript, this is the default value of any color.

BLANCHED_ALMOND = `Color(1, 0.921569, 0.803922, 1)`

Blanched almond color.

BLUE = `Color(0, 0, 1, 1)`

Blue color.

BLUE_VIOLET = `Color(0.541176, 0.168627, 0.886275, 1)`

Blue violet color.

BROWN = `Color(0.647059, 0.164706, 0.164706, 1)`

Brown color.

BURLYWOOD = `Color(0.870588, 0.721569, 0.529412, 1)`

Burlywood color.

CADET_BLUE = `Color(0.372549, 0.619608, 0.627451, 1)`

Cadet blue color.

CHARTREUSE = `Color(0.498039, 1, 0, 1)`

Chartreuse color.

CHOCOLATE = `Color(0.823529, 0.411765, 0.117647, 1)`

Chocolate color.

CORAL = `Color(1, 0.498039, 0.313726, 1)`

Coral color.

CORNFLOWER_BLUE = `Color(0.392157, 0.584314, 0.929412, 1)`

Cornflower blue color.

CORNSILK = `Color(1, 0.972549, 0.862745, 1)`

Cornsilk color.

CRIMSON = `Color(0.862745, 0.0784314, 0.235294, 1)`

Crimson color.

CYAN = `Color(0, 1, 1, 1)`

Cyan color.

DARK_BLUE = `Color(0, 0, 0.545098, 1)`

Dark blue color.

DARK_CYAN = `Color(0, 0.545098, 0.545098, 1)`

Dark cyan color.

DARK_GOLDENROD = `Color(0.721569, 0.52549, 0.0431373, 1)`

Dark goldenrod color.

DARK_GRAY = `Color(0.662745, 0.662745, 0.662745, 1)`

Dark gray color.

DARK_GREEN = `Color(0, 0.392157, 0, 1)`

Dark green color.

DARK_KHAKI = `Color(0.741176, 0.717647, 0.419608, 1)`

Dark khaki color.

DARK_MAGENTA = `Color(0.545098, 0, 0.545098, 1)`

Dark magenta color.

DARK_OLIVE_GREEN = `Color(0.333333, 0.419608, 0.184314, 1)`

Dark olive green color.

DARK_ORANGE = `Color(1, 0.54902, 0, 1)`

Dark orange color.

DARK_ORCHID = `Color(0.6, 0.196078, 0.8, 1)`

Dark orchid color.

DARK_RED = `Color(0.545098, 0, 0, 1)`

Dark red color.

DARK_SALMON = `Color(0.913725, 0.588235, 0.478431, 1)`

Dark salmon color.

DARK_SEA_GREEN = `Color(0.560784, 0.737255, 0.560784, 1)`

Dark sea green color.

DARK_SLATE_BLUE = `Color(0.282353, 0.239216, 0.545098, 1)`

Dark slate blue color.

DARK_SLATE_GRAY = `Color(0.184314, 0.309804, 0.309804, 1)`

Dark slate gray color.

DARK_TURQUOISE = `Color(0, 0.807843, 0.819608, 1)`

Dark turquoise color.

DARK_VIOLET = `Color(0.580392, 0, 0.827451, 1)`

Dark violet color.

DEEP_PINK = `Color(1, 0.0784314, 0.576471, 1)`

Deep pink color.

DEEP_SKY_BLUE = `Color(0, 0.74902, 1, 1)`

Deep sky blue color.

DIM_GRAY = `Color(0.411765, 0.411765, 0.411765, 1)`

Dim gray color.

DODGER_BLUE = `Color(0.117647, 0.564706, 1, 1)`

Dodger blue color.

FIREBRICK = `Color(0.698039, 0.133333, 0.133333, 1)`

Firebrick color.

FLORAL_WHITE = `Color(1, 0.980392, 0.941176, 1)`

Floral white color.

FOREST_GREEN = `Color(0.133333, 0.545098, 0.133333, 1)`

Forest green color.

FUCHSIA = `Color(1, 0, 1, 1)`

Fuchsia color.

GAINSBORO = `Color(0.862745, 0.862745, 0.862745, 1)`

Gainsboro color.

GHOST_WHITE = `Color(0.972549, 0.972549, 1, 1)`

Ghost white color.

GOLD = `Color(1, 0.843137, 0, 1)`

Gold color.

GOLDENROD = `Color(0.854902, 0.647059, 0.12549, 1)`

Goldenrod color.

GRAY = `Color(0.745098, 0.745098, 0.745098, 1)`

Gray color.

GREEN = `Color(0, 1, 0, 1)`

Green color.

GREEN_YELLOW = `Color(0.678431, 1, 0.184314, 1)`

Green yellow color.

HONEYDEW = `Color(0.941176, 1, 0.941176, 1)`

Honeydew color.

HOT_PINK = `Color(1, 0.411765, 0.705882, 1)`

Hot pink color.

INDIAN_RED = `Color(0.803922, 0.360784, 0.360784, 1)`

Indian red color.

INDIGO = `Color(0.294118, 0, 0.509804, 1)`

Indigo color.

IVORY = `Color(1, 1, 0.941176, 1)`

Ivory color.

KHAKI = `Color(0.941176, 0.901961, 0.54902, 1)`

Khaki color.

LAVENDER = `Color(0.901961, 0.901961, 0.980392, 1)`

Lavender color.

LAVENDER_BLUSH = `Color(1, 0.941176, 0.960784, 1)`

Lavender blush color.

LAWN_GREEN = `Color(0.486275, 0.988235, 0, 1)`

Lawn green color.

LEMON_CHIFFON = `Color(1, 0.980392, 0.803922, 1)`

Lemon chiffon color.

LIGHT_BLUE = `Color(0.678431, 0.847059, 0.901961, 1)`

Light blue color.

LIGHT_CORAL = `Color(0.941176, 0.501961, 0.501961, 1)`

Light coral color.

LIGHT_CYAN = `Color(0.878431, 1, 1, 1)`

Light cyan color.

LIGHT_GOLDENROD = `Color(0.980392, 0.980392, 0.823529, 1)`

Light goldenrod color.

LIGHT_GRAY = `Color(0.827451, 0.827451, 0.827451, 1)`

Light gray color.

LIGHT_GREEN = `Color(0.564706, 0.933333, 0.564706, 1)`

Light green color.

LIGHT_PINK = `Color(1, 0.713726, 0.756863, 1)`

Light pink color.

LIGHT_SALMON = `Color(1, 0.627451, 0.478431, 1)`

Light salmon color.

LIGHT_SEA_GREEN = `Color(0.12549, 0.698039, 0.666667, 1)`

Light sea green color.

LIGHT_SKY_BLUE = `Color(0.529412, 0.807843, 0.980392, 1)`

Light sky blue color.

LIGHT_SLATE_GRAY = `Color(0.466667, 0.533333, 0.6, 1)`

Light slate gray color.

LIGHT_STEEL_BLUE = `Color(0.690196, 0.768627, 0.870588, 1)`

Light steel blue color.

LIGHT_YELLOW = `Color(1, 1, 0.878431, 1)`

Light yellow color.

LIME = `Color(0, 1, 0, 1)`

Lime color.

LIME_GREEN = `Color(0.196078, 0.803922, 0.196078, 1)`

Lime green color.

LINEN = `Color(0.980392, 0.941176, 0.901961, 1)`

Linen color.

MAGENTA = `Color(1, 0, 1, 1)`

Magenta color.

MAROON = `Color(0.690196, 0.188235, 0.376471, 1)`

Maroon color.

MEDIUM_AQUAMARINE = `Color(0.4, 0.803922, 0.666667, 1)`

Medium aquamarine color.

MEDIUM_BLUE = `Color(0, 0, 0.803922, 1)`

Medium blue color.

MEDIUM_ORCHID = `Color(0.729412, 0.333333, 0.827451, 1)`

Medium orchid color.

MEDIUM_PURPLE = `Color(0.576471, 0.439216, 0.858824, 1)`

Medium purple color.

MEDIUM_SEA_GREEN = `Color(0.235294, 0.701961, 0.443137, 1)`

Medium sea green color.

MEDIUM_SLATE_BLUE = `Color(0.482353, 0.407843, 0.933333, 1)`

Medium slate blue color.

MEDIUM_SPRING_GREEN = `Color(0, 0.980392, 0.603922, 1)`

Medium spring green color.

MEDIUM_TURQUOISE = `Color(0.282353, 0.819608, 0.8, 1)`

Medium turquoise color.

MEDIUM_VIOLET_RED = `Color(0.780392, 0.0823529, 0.521569, 1)`

Medium violet red color.

MIDNIGHT_BLUE = `Color(0.0980392, 0.0980392, 0.439216, 1)`

Midnight blue color.

MINT_CREAM = `Color(0.960784, 1, 0.980392, 1)`

Mint cream color.

MISTY_ROSE = `Color(1, 0.894118, 0.882353, 1)`

Misty rose color.

MOCCASIN = `Color(1, 0.894118, 0.709804, 1)`

Moccasin color.

NAVAJO_WHITE = `Color(1, 0.870588, 0.678431, 1)`

Navajo white color.

NAVY_BLUE = `Color(0, 0, 0.501961, 1)`

Navy blue color.

OLD_LACE = `Color(0.992157, 0.960784, 0.901961, 1)`

Old lace color.

OLIVE = `Color(0.501961, 0.501961, 0, 1)`

Olive color.

OLIVE_DRAB = `Color(0.419608, 0.556863, 0.137255, 1)` 🔗

Olive drab color.

ORANGE = `Color(1, 0.647059, 0, 1)` 🔗

Orange color.

ORANGE_RED = `Color(1, 0.270588, 0, 1)` 🔗

Orange red color.

ORCHID = `Color(0.854902, 0.439216, 0.839216, 1)` 🔗

Orchid color.

PALE_GOLDENROD = `Color(0.933333, 0.909804, 0.666667, 1)` 🔗

Pale goldenrod color.

PALE_GREEN = `Color(0.596078, 0.984314, 0.596078, 1)` 🔗

Pale green color.

PALE_TURQUOISE = `Color(0.686275, 0.933333, 0.933333, 1)` 🔗

Pale turquoise color.

PALE_VIOLET_RED = `Color(0.858824, 0.439216, 0.576471, 1)` 🔗

Pale violet red color.

PAPAYA_WHIP = `Color(1, 0.937255, 0.835294, 1)`

Papaya whip color.

PEACH_PUFF = `Color(1, 0.854902, 0.72549, 1)`

Peach puff color.

PERU = `Color(0.803922, 0.521569, 0.247059, 1)`

Peru color.

PINK = `Color(1, 0.752941, 0.796078, 1)`

Pink color.

PLUM = `Color(0.866667, 0.627451, 0.866667, 1)`

Plum color.

POWDER_BLUE = `Color(0.690196, 0.878431, 0.901961, 1)`

Powder blue color.

PURPLE = `Color(0.627451, 0.12549, 0.941176, 1)`

Purple color.

REBECCA_PURPLE = `Color(0.4, 0.2, 0.6, 1)`

Rebecca purple color.

RED = `Color(1, 0, 0, 1)`

Red color.

ROSY_BROWN = `Color(0.737255, 0.560784, 0.560784, 1)`

Rosy brown color.

ROYAL_BLUE = `Color(0.254902, 0.411765, 0.882353, 1)`

Royal blue color.

SADDLE_BROWN = `Color(0.545098, 0.270588, 0.0745098, 1)`

Saddle brown color.

SALMON = `Color(0.980392, 0.501961, 0.447059, 1)`

Salmon color.

SANDY_BROWN = `Color(0.956863, 0.643137, 0.376471, 1)`

Sandy brown color.

SEA_GREEN = `Color(0.180392, 0.545098, 0.341176, 1)`

Sea green color.

SEASHELL = `Color(1, 0.960784, 0.933333, 1)`

Seashell color.

SIENNA = `Color(0.627451, 0.321569, 0.176471, 1)`

Sienna color.

SILVER = `Color(0.752941, 0.752941, 0.752941, 1)`

Silver color.

SKY_BLUE = `Color(0.529412, 0.807843, 0.921569, 1)`

Sky blue color.

SLATE_BLUE = `Color(0.415686, 0.352941, 0.803922, 1)`

Slate blue color.

SLATE_GRAY = `Color(0.439216, 0.501961, 0.564706, 1)`

Slate gray color.

SNOW = `Color(1, 0.980392, 0.980392, 1)`

Snow color.

SPRING_GREEN = `Color(0, 1, 0.498039, 1)`

Spring green color.

STEEL_BLUE = `Color(0.27451, 0.509804, 0.705882, 1)`

Steel blue color.

TAN = `Color(0.823529, 0.705882, 0.54902, 1)`

Tan color.

TEAL = `Color(0, 0.501961, 0.501961, 1)`

Teal color.

THISTLE = `Color(0.847059, 0.74902, 0.847059, 1)`

Thistle color.

TOMATO = `Color(1, 0.388235, 0.278431, 1)`

Tomato color.

TRANSPARENT = `Color(1, 1, 1, 0)`

Transparent color (white with zero alpha).

TURQUOISE = `Color(0.25098, 0.878431, 0.815686, 1)`

Turquoise color.

VIOLET = `Color(0.933333, 0.509804, 0.933333, 1)`

Violet color.

WEB_GRAY = `Color(0.501961, 0.501961, 0.501961, 1)`

Web gray color.

WEB_GREEN = `Color(0, 0.501961, 0, 1)`

Web green color.

WEB_MAROON = `Color(0.501961, 0, 0, 1)`

Web maroon color.

WEB_PURPLE = `Color(0.501961, 0, 0.501961, 1)`

Web purple color.

WHEAT = `Color(0.960784, 0.870588, 0.701961, 1)`

Wheat color.

WHITE = `Color(1, 1, 1, 1)`

White color.

WHITE_SMOKE = `Color(0.960784, 0.960784, 0.960784, 1)`

White smoke color.

YELLOW = `Color(1, 1, 0, 1)`

Yellow color.

YELLOW_GREEN = `Color(0.603922, 0.803922, 0.196078, 1)`

Yellow green color.

Property Descriptions

float **a** = `1.0`

The color's alpha component, typically on the range of 0 to 1. A value of 0 means that the color is fully transparent. A value of 1 means that the color is fully opaque.

---

int **a8** = `255`

Wrapper for a that uses the range 0 to 255, instead of 0 to 1.

---

float **b** = `0.0`

The color's blue component, typically on the range of 0 to 1.

---

int **b8** = `0`

Wrapper for b that uses the range 0 to 255, instead of 0 to 1.

---

float **g** = `0.0`

The color's green component, typically on the range of 0 to 1.

int **g8** = `0`

Wrapper for g that uses the range 0 to 255, instead of 0 to 1.

float **h** = `0.0`

The HSV hue of this color, on the range 0 to 1.

float **r** = `0.0`

The color's red component, typically on the range of 0 to 1.

int **r8** = `0`

Wrapper for r that uses the range 0 to 255, instead of 0 to 1.

float **s** = `0.0`

The HSV saturation of this color, on the range 0 to 1.

float **v** = `0.0`

The HSV value (brightness) of this color, on the range 0 to 1.

Constructor Descriptions

Color **Color**()

Constructs a default **Color** from opaque black. This is the same as BLACK.

Note: in C#, constructs an empty color with all of its components set to `0.0` (transparent black).

Color **Color**(from: Color, alpha: float)

Constructs a **Color** from the existing color, with a set to the given `alpha` value.

GDScript

```
var red = Color(Color.RED, 0.2) # 20% opaque red.
```

C#

```
var red = new Color(Colors.Red, 0.2f); // 20% opaque red.
```

Color **Color**(from: Color)

Constructs a **Color** as a copy of the given **Color**.

Color **Color**(code: String)

Constructs a **Color** either from an HTML color code or from a standardized color name. The supported color names are the same as the constants.

Color **Color**(code: String, alpha: float)

Constructs a **Color** either from an HTML color code or from a standardized color name, with `alpha` on the range of 0.0 to 1.0. The supported color names are the same as the constants.

Color **Color**(r: float, g: float, b: float)

Constructs a **Color** from RGB values, typically between 0.0 and 1.0. a is set to 1.0.

GDScript

```
var color = Color(0.2, 1.0, 0.7) # Similar to `Color8(51, 255,
178, 255)`
```

C#

```
var color = new Color(0.2f, 1.0f, 0.7f); // Similar to
`Color.Color8(51, 255, 178, 255)`
```

Color **Color**(r: float, g: float, b: float, a: float)

Constructs a **Color** from RGBA values, typically between 0.0 and 1.0.

GDScript

```
var color = Color(0.2, 1.0, 0.7, 0.8) # Similar to `Color8(51,
255, 178, 204)`
```

C#

```
var color = new Color(0.2f, 1.0f, 0.7f, 0.8f); // Similar to
`Color.Color8(51, 255, 178, 255, 204)`
```

Method Descriptions

Color **blend**(over: Color) const 🔗

Returns a new color resulting from overlaying this color over the given color. In a painting program, you can imagine it as the `over` color painted over this color (including alpha).

GDScript

```
var bg = Color(0.0, 1.0, 0.0, 0.5) # Green with alpha of 50%
var fg = Color(1.0, 0.0, 0.0, 0.5) # Red with alpha of 50%
var blended_color = bg.blend(fg) # Brown with alpha of 75%
```

C#

```
var bg = new Color(0.0f, 1.0f, 0.0f, 0.5f); // Green with alpha of 50%
var fg = new Color(1.0f, 0.0f, 0.0f, 0.5f); // Red with alpha of 50%
Color blendedColor = bg.Blend(fg); // Brown with alpha of 75%
```

Color **clamp**(min: Color = Color(0, 0, 0, 0), max: Color = Color(1, 1, 1, 1)) const 🔗

Returns a new color with all components clamped between the components of `min` and `max`, by running @GlobalScope.clamp on each component.

Color **darkened**(amount: float) const 🔗

Returns a new color resulting from making this color darker by the specified `amount` (ratio from 0.0 to 1.0). See also lightened.

GDScript

```
var green = Color(0.0, 1.0, 0.0)
var darkgreen = green.darkened(0.2) # 20% darker than regular green
```

C#

```
var green = new Color(0.0f, 1.0f, 0.0f);
Color darkgreen = green.Darkened(0.2f); // 20% darker than
regular green
```

Color **from_hsv**(h: float, s: float, v: float, alpha: float = 1.0) static 🔗

Constructs a color from an HSV profile [https://en.wikipedia.org/wiki/HSL_and_HSV]. The hue (`h`), saturation (`s`), and value (`v`) are typically between 0.0 and 1.0.

GDScript

```
var color = Color.from_hsv(0.58, 0.5, 0.79, 0.8)
```

C#

```
var color = Color.FromHsv(0.58f, 0.5f, 0.79f, 0.8f);
```

Color **from_ok_hsl**(h: float, s: float, l: float, alpha: float = 1.0) static 🔗

Constructs a color from an OK HSL profile [https://bottosson.github.io/posts/colorpicker/]. The hue (`h`), saturation (`s`), and lightness (`l`) are typically between 0.0 and 1.0.

GDScript

```
var color = Color.from_ok_hsl(0.58, 0.5, 0.79, 0.8)
```

C#

```
var color = Color.FromOkHsl(0.58f, 0.5f, 0.79f, 0.8f);
```

Color **from_rgbe9995**(rgbe: int) static 🔗

Decodes a **Color** from an RGBE9995 format integer. See Image.FORMAT_RGBE9995.

Color **from_string**(str: String, default: Color) static 🔗

Creates a **Color** from the given string, which can be either an HTML color code or a named color (case-insensitive). Returns `default` if the color cannot be inferred from the string.

If you want to create a color from String in a constant expression, use the equivalent constructor instead (i.e. `Color("color string")`).

---

float **get_luminance**() const 🔗

Returns the light intensity of the color, as a value between 0.0 and 1.0 (inclusive). This is useful when determining light or dark color. Colors with a luminance smaller than 0.5 can be generally considered dark.

Note: get_luminance relies on the color being in the linear color space to return an accurate relative luminance value. If the color is in the sRGB color space, use srgb_to_linear to convert it to the linear color space first.

Color **hex**(hex: int) static 🔗

Returns the **Color** associated with the provided `hex` integer in 32-bit RGBA format (8 bits per channel). This method is the inverse of to_rgba32.

In GDScript and C#, the int is best visualized with hexadecimal notation (`"0x"` prefix, making it `"0xRRGGBBAA"`).

GDScript

```
var red = Color.hex(0xff0000ff)
var dark_cyan = Color.hex(0x008b8bff)
var my_color = Color.hex(0xbbefd2a4)
```

C#

```
var red = new Color(0xff0000ff);
var dark_cyan = new Color(0x008b8bff);
var my_color = new Color(0xbbefd2a4);
```

If you want to use hex notation in a constant expression, use the equivalent constructor instead (i.e. `Color(0xRRGGBBAA)`).

Color **hex64**(hex: int) static 🔗

Returns the **Color** associated with the provided `hex` integer in 64-bit RGBA format (16 bits per channel). This method is the inverse of to_rgba64.

In GDScript and C#, the int is best visualized with hexadecimal notation (`"0x"` prefix, making it `"0xRRRRGGGGBBBBAAAA"`).

Color **html**(rgba: String) static 🔗

Returns a new color from `rgba`, an HTML hexadecimal color string. `rgba` is not case-sensitive, and may be prefixed by a hash sign (`#`).

`rgba` must be a valid three-digit or six-digit hexadecimal color string, and may contain an alpha channel value. If `rgba` does not contain an alpha channel value, an alpha channel value of 1.0 is applied. If `rgba` is invalid, returns an empty color.

GDScript

```
var blue = Color.html("#0000ff") # blue is Color(0.0, 0.0, 1.0,
1.0)
```

```
var green = Color.html("#0F0")    # green is Color(0.0, 1.0,
0.0, 1.0)
var col = Color.html("663399cc") # col is Color(0.4, 0.2, 0.6,
0.8)
```

C#

```
var blue = Color.FromHtml("#0000ff"); // blue is Color(0.0,
0.0, 1.0, 1.0)
var green = Color.FromHtml("#0F0");   // green is Color(0.0,
1.0, 0.0, 1.0)
var col = Color.FromHtml("663399cc"); // col is Color(0.4, 0.2,
0.6, 0.8)
```

---

bool **html_is_valid**(color: String) static 🔗

Returns `true` if `color` is a valid HTML hexadecimal color string. The string must be a hexadecimal value (case-insensitive) of either 3, 4, 6 or 8 digits, and may be prefixed by a hash sign (`#`). This method is identical to String.is_valid_html_color.

GDScript

```
Color.html_is_valid("#55aaFF")   # Returns true
Color.html_is_valid("#55AAFF20") # Returns true
Color.html_is_valid("55AAFF")    # Returns true
Color.html_is_valid("#F2C")      # Returns true

Color.html_is_valid("#AABBC")    # Returns false
Color.html_is_valid("#55aaFF5")  # Returns false
```

C#

```
Color.HtmlIsValid("#55AAFF");   // Returns true
Color.HtmlIsValid("#55AAFF20"); // Returns true
Color.HtmlIsValid("55AAFF");    // Returns true
Color.HtmlIsValid("#F2C");      // Returns true
Color.HtmlIsValid("#AABBC");     // Returns false
Color.HtmlIsValid("#55aaFF5");   // Returns false
```

Color **inverted**() const 🔗

Returns the color with its r, g, and b components inverted (`(1 - r, 1 - g, 1 - b, a)`).

GDScript

```
var black = Color.WHITE.inverted()
var color = Color(0.3, 0.4, 0.9)
var inverted_color = color.inverted() # Equivalent to
`Color(0.7, 0.6, 0.1)`
```

C#

```
var black = Colors.White.Inverted();
var color = new Color(0.3f, 0.4f, 0.9f);
Color invertedColor = color.Inverted(); // Equivalent to `new
Color(0.7f, 0.6f, 0.1f)`
```

---

bool **is_equal_approx**(to: Color) const 🔗

Returns `true` if this color and `to` are approximately equal, by running @GlobalScope.is_equal_approx on each component.

Color **lerp**(to: Color, weight: float) const 🔗

Returns the linear interpolation between this color's components and `to`'s components. The interpolation factor `weight` should be between 0.0 and 1.0 (inclusive). See also @GlobalScope.lerp.

GDScript

```
var red = Color(1.0, 0.0, 0.0)
var aqua = Color(0.0, 1.0, 0.8)

red.lerp(aqua, 0.2) # Returns Color(0.8, 0.2, 0.16)
red.lerp(aqua, 0.5) # Returns Color(0.5, 0.5, 0.4)
red.lerp(aqua, 1.0) # Returns Color(0.0, 1.0, 0.8)
```

C#

```
var red = new Color(1.0f, 0.0f, 0.0f);
var aqua = new Color(0.0f, 1.0f, 0.8f);

red.Lerp(aqua, 0.2f); // Returns Color(0.8f, 0.2f, 0.16f)
red.Lerp(aqua, 0.5f); // Returns Color(0.5f, 0.5f, 0.4f)
red.Lerp(aqua, 1.0f); // Returns Color(0.0f, 1.0f, 0.8f)
```

Color **lightened**(amount: float) const 🔗

Returns a new color resulting from making this color lighter by the specified `amount`, which should be a ratio from 0.0 to 1.0. See also darkened.

GDScript

```
var green = Color(0.0, 1.0, 0.0)
var light_green = green.lightened(0.2) # 20% lighter than regular green
```

C#

```
var green = new Color(0.0f, 1.0f, 0.0f);
Color lightGreen = green.Lightened(0.2f); // 20% lighter than regular green
```

Color **linear_to_srgb**() const 🔗

Returns the color converted to the sRGB [https://en.wikipedia.org/wiki/SRGB] color space. This method assumes the original color is in the linear color space. See also srgb_to_linear which performs the opposite operation.

Color **srgb_to_linear**() const

Returns the color converted to the linear color space. This method assumes the original color already is in the sRGB color space. See also linear_to_srgb which performs the opposite operation.

int **to_abgr32**() const

Returns the color converted to a 32-bit integer in ABGR format (each component is 8 bits). ABGR is the reversed version of the default RGBA format.

GDScript

```
var color = Color(1, 0.5, 0.2)
print(color.to_abgr32()) # Prints 4281565439
```

C#

```
var color = new Color(1.0f, 0.5f, 0.2f);
GD.Print(color.ToAbgr32()); // Prints 4281565439
```

int **to_abgr64**() const

Returns the color converted to a 64-bit integer in ABGR format (each component is 16 bits). ABGR is the reversed version of the default RGBA format.

GDScript

```
var color = Color(1, 0.5, 0.2)
```

```
print(color.to_abgr64()) # Prints -225178692812801
```

C#

```
var color = new Color(1.0f, 0.5f, 0.2f);
GD.Print(color.ToAbgr64()); // Prints -225178692812801
```

---

int **to_argb32**() const

Returns the color converted to a 32-bit integer in ARGB format (each component is 8 bits). ARGB is more compatible with DirectX.

GDScript

```
var color = Color(1, 0.5, 0.2)
print(color.to_argb32()) # Prints 4294934323
```

C#

```
var color = new Color(1.0f, 0.5f, 0.2f);
GD.Print(color.ToArgb32()); // Prints 4294934323
```

---

int **to_argb64**() const

Returns the color converted to a 64-bit integer in ARGB format (each component is 16 bits). ARGB is more compatible with DirectX.

GDScript

```
var color = Color(1, 0.5, 0.2)
print(color.to_argb64()) # Prints -2147470541
```

C#

```
var color = new Color(1.0f, 0.5f, 0.2f);
GD.Print(color.ToArgb64()); // Prints -2147470541
```

String **to_html**(with_alpha: bool = true) const

Returns the color converted to an HTML hexadecimal color String in RGBA format, without the hash (`#`) prefix.

Setting `with_alpha` to `false`, excludes alpha from the hexadecimal string, using RGB format instead of RGBA format.

GDScript

```
var white = Color(1, 1, 1, 0.5)
var with_alpha = white.to_html() # Returns "ffffff7f"
var without_alpha = white.to_html(false) # Returns "ffffff"
```

C#

```
var white = new Color(1, 1, 1, 0.5f);
string withAlpha = white.ToHtml(); // Returns "ffffff7f"
string withoutAlpha = white.ToHtml(false); // Returns "ffffff"
```

---

int **to_rgba32**() const

Returns the color converted to a 32-bit integer in RGBA format (each component is 8 bits). RGBA is Godot's default format. This method is the inverse of hex.

GDScript

```
var color = Color(1, 0.5, 0.2)
print(color.to_rgba32()) # Prints 4286526463
```

C#

```
var color = new Color(1, 0.5f, 0.2f);
GD.Print(color.ToRgba32()); // Prints 4286526463
```

---

int **to_rgba64**() const

Returns the color converted to a 64-bit integer in RGBA format (each component is 16 bits). RGBA is Godot's default format. This method is the inverse of hex64.

GDScript

```
var color = Color(1, 0.5, 0.2)
print(color.to_rgba64()) # Prints -140736629309441
```

C#

```
var color = new Color(1, 0.5f, 0.2f);
GD.Print(color.ToRgba64()); // Prints -140736629309441
```

Operator Descriptions

bool **operator !=**(right: Color)

Returns `true` if the colors are not exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Color **operator ***(right: Color)

Multiplies each component of the **Color** by the components of the given **Color**.

Color **operator ***(right: float)

Multiplies each component of the **Color** by the given float.

Color **operator ***(right: int)

Multiplies each component of the **Color** by the given int.

Color **operator +**(right: Color)

Adds each component of the **Color** with the components of the given **Color**.

Color **operator -**(right: Color)

Subtracts each component of the **Color** by the components of the given **Color**.

Color **operator /**(right: Color)

Divides each component of the **Color** by the components of the given **Color**.

Color **operator /**(right: float)

Divides each component of the **Color** by the given float.

Color **operator /**(right: int)

Divides each component of the **Color** by the given int.

---

bool **operator ==**(right: Color)

Returns `true` if the colors are exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

---

float **operator []**(index: int) 🔗

Access color components using their index. `[0]` is equivalent to r, `[1]` is equivalent to g, `[2]` is equivalent to b, and `[3]` is equivalent to a.

Color **operator unary+**() 🔗

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

Color **operator unary-**() 🔗

Inverts the given color. This is equivalent to `Color.WHITE - c` or `Color(1 - c.r, 1 - c.g, 1 - c.b, 1 - c.a)`. Unlike with inverted, the a component is inverted, too.

Dictionary

A built-in data structure that holds key-value pairs.

Description

Dictionaries are associative containers that contain values referenced by unique keys. Dictionaries will preserve the insertion order when adding new entries. In other programming languages, this data structure is often referred to as a hash map or an associative array.

You can define a dictionary by placing a comma-separated list of `key: value` pairs inside curly braces `{}`.

Creating a dictionary:

GDScript

```
var my_dict = {} # Creates an empty dictionary.

var dict_variable_key = "Another key name"
var dict_variable_value = "value2"
var another_dict = {
    "Some key name": "value1",
    dict_variable_key: dict_variable_value,
}

var points_dict = {"White": 50, "Yellow": 75, "Orange": 100}

# Alternative Lua-style syntax.
# Doesn't require quotes around keys, but only string constants
can be used as key names.
# Additionally, key names must start with a letter or an
underscore.
# Here, `some_key` is a string literal, not a variable!
another_dict = {
    some_key = 42,
}
```

C#

```
var myDict = new Godot.Collections.Dictionary(); // Creates an empty dictionary.
var pointsDict = new Godot.Collections.Dictionary
{
    {"White", 50},
    {"Yellow", 75},
    {"Orange", 100}
};
```

You can access a dictionary's value by referencing its corresponding key. In the above example, `points_dict["White"]` will return `50`. You can also write `points_dict.White`, which is equivalent. However, you'll have to use the bracket syntax if the key you're accessing the dictionary with isn't a fixed string (such as a number or variable).

GDScript

```
@export_enum("White", "Yellow", "Orange") var my_color: String
var points_dict = {"White": 50, "Yellow": 75, "Orange": 100}
func _ready():
    # We can't use dot syntax here as `my_color` is a variable.
    var points = points_dict[my_color]
```

C#

```
[Export(PropertyHint.Enum, "White,Yellow,Orange")]
public string MyColor { get; set; }
private Godot.Collections.Dictionary _pointsDict = new
Godot.Collections.Dictionary
{
    {"White", 50},
    {"Yellow", 75},
    {"Orange", 100}
};
```

```
public override void _Ready()
{
    int points = (int)_pointsDict[MyColor];
}
```

In the above code, `points` will be assigned the value that is paired with the appropriate color selected in `my_color`.

Dictionaries can contain more complex data:

GDScript

```
var my_dict = {
    "First Array": [1, 2, 3, 4] # Assigns an Array to a String key.
}
```

C#

```
var myDict = new Godot.Collections.Dictionary
{
    {"First Array", new Godot.Collections.Array{1, 2, 3, 4}}
};
```

To add a key to an existing dictionary, access it like an existing key and assign to it:

GDScript

```
var points_dict = {"White": 50, "Yellow": 75, "Orange": 100}
points_dict["Blue"] = 150 # Add "Blue" as a key and assign 150 as its value.
```

C#

```
var pointsDict = new Godot.Collections.Dictionary
{
    {"White", 50},
    {"Yellow", 75},
    {"Orange", 100}
};
pointsDict["Blue"] = 150; // Add "Blue" as a key and assign 150
as its value.
```

Finally, dictionaries can contain different types of keys and values in the same dictionary:

GDScript

```
# This is a valid dictionary.
# To access the string "Nested value" below, use
`my_dict.sub_dict.sub_key` or `my_dict["sub_dict"]["sub_key"]`.
# Indexing styles can be mixed and matched depending on your
needs.
var my_dict = {
    "String Key": 5,
    4: [1, 2, 3],
    7: "Hello",
    "sub_dict": {"sub_key": "Nested value"},
}
```

C#

```
// This is a valid dictionary.
// To access the string "Nested value" below, use
`((Godot.Collections.Dictionary)myDict["sub_dict"])
["sub_key"]`.
var myDict = new Godot.Collections.Dictionary {
    {"String Key", 5},
    {4, new Godot.Collections.Array{1,2,3}},
    {7, "Hello"},
    {"sub_dict", new Godot.Collections.Dictionary{{"sub_key",
"Nested value"}}}
};
```

The keys of a dictionary can be iterated with the `for` keyword:

GDScript

```
var groceries = {"Orange": 20, "Apple": 2, "Banana": 4}
for fruit in groceries:
    var amount = groceries[fruit]
```

C#

```
var groceries = new Godot.Collections.Dictionary{{"Orange", 
20}, {"Apple", 2}, {"Banana", 4}};
foreach (var (fruit, amount) in groceries)
{
    // `fruit` is the key, `amount` is the value.
}
```

Note: Dictionaries are always passed by reference. To get a copy of a dictionary which can be modified independently of the original dictionary, use duplicate.

Note: Erasing elements while iterating over dictionaries is **not** supported and will result in unpredictable behavior.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- GDScript basics: Dictionary
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]
- Operating System Testing Demo [https://godotengine.org/asset-library/asset/2789]

Constructors

Dictionary Dictionary()

Dictionary	Dictionary(from: Dictionary)

Methods

void	clear()
Dictionary	duplicate(deep: bool = false) const
bool	erase(key: Variant)
Variant	find_key(value: Variant) const
Variant	get(key: Variant, default: Variant = null) const
Variant	get_or_add(key: Variant, default: Variant = null)
bool	has(key: Variant) const
bool	has_all(keys: Array) const
int	hash() const
bool	is_empty() const

bool	is_read_only() const
Array	keys() const
void	make_read_only()
void	merge(dictionary: Dictionary, overwrite: bool = false)
Dictionary	merged(dictionary: Dictionary, overwrite: bool = false) const
bool	recursive_equal(dictionary: Dictionary, recursion_count: int) const
int	size() const
Array	values() const

Operators

bool	operator !=(right: Dictionary)
bool	operator ==(right: Dictionary)
Variant	operator [](key: Variant)

Constructor Descriptions

Dictionary **Dictionary**() 🔗

Constructs an empty **Dictionary**.

Dictionary **Dictionary**(from: Dictionary)

Returns the same dictionary as `from`. If you need a copy of the dictionary, use duplicate.

Method Descriptions

void **clear**() 🔗

Clears the dictionary, removing all entries from it.

Dictionary **duplicate**(deep: bool = false) const 🔗

Creates and returns a new copy of the dictionary. If `deep` is `true`, inner **Dictionary** and Array keys and values are also copied, recursively.

---

bool **erase**(key: Variant) 🔗

Removes the dictionary entry by key, if it exists. Returns `true` if the given `key` existed in the dictionary, otherwise `false`.

Note: Do not erase entries while iterating over the dictionary. You can iterate over the keys array instead.

Variant **find_key**(value: Variant) const

Finds and returns the first key whose associated value is equal to `value`, or `null` if it is not found.

Note: `null` is also a valid key. If inside the dictionary, find_key may give misleading results.

Variant **get**(key: Variant, default: Variant = null) const

Returns the corresponding value for the given `key` in the dictionary. If the `key` does not exist, returns `default`, or `null` if the parameter is omitted.

Variant **get_or_add**(key: Variant, default: Variant = null)

Gets a value and ensures the key is set. If the `key` exists in the dictionary, this behaves like get. Otherwise, the `default` value is inserted into the dictionary and returned.

bool **has**(key: Variant) const

Returns `true` if the dictionary contains an entry with the given `key`.

GDScript

```
var my_dict = {
    "Godot" : 4,
    210 : null,
}
```

```
print(my_dict.has("Godot")) # Prints true
print(my_dict.has(210))     # Prints true
print(my_dict.has(4))       # Prints false
```

C#

```
var myDict = new Godot.Collections.Dictionary
{
    { "Godot", 4 },
    { 210, default },
};

GD.Print(myDict.ContainsKey("Godot")); // Prints true
GD.Print(myDict.ContainsKey(210));     // Prints true
GD.Print(myDict.ContainsKey(4));       // Prints false
```

In GDScript, this is equivalent to the `in` operator:

```
if "Godot" in {"Godot": 4}:
    print("The key is here!") # Will be printed.
```

Note: This method returns `true` as long as the `key` exists, even if its corresponding value is `null`.

---

bool **has_all**(keys: Array) const

Returns `true` if the dictionary contains all keys in the given `keys` array.

```
var data = {"width" : 10, "height" : 20}
data.has_all(["height", "width"]) # Returns true
```

---

int **hash**() const

Returns a hashed 32-bit integer value representing the dictionary contents.

GDScript

```
var dict1 = {"A": 10, "B": 2}
var dict2 = {"A": 10, "B": 2}

print(dict1.hash() == dict2.hash()) # Prints true
```

C#

```
var dict1 = new Godot.Collections.Dictionary{{"A", 10}, {"B",
2}};
var dict2 = new Godot.Collections.Dictionary{{"A", 10}, {"B",
2}};

// Godot.Collections.Dictionary has no Hash() method. Use
GD.Hash() instead.
GD.Print(GD.Hash(dict1) == GD.Hash(dict2)); // Prints true
```

Note: Dictionaries with the same entries but in a different order will not have the same hash.

Note: Dictionaries with equal hash values are *not* guaranteed to be the same, because of hash collisions. On the contrary, dictionaries with different hash values are guaranteed to be different.

---

bool **is_empty**() const

Returns `true` if the dictionary is empty (its size is `0`). See also size.

---

bool **is_read_only**() const

Returns `true` if the dictionary is read-only. See make_read_only. Dictionaries are automatically read-only if declared with `const` keyword.

Array **keys**() const

Returns the list of keys in the dictionary.

---

void **make_read_only**() 🔗

Makes the dictionary read-only, i.e. disables modification of the dictionary's contents. Does not apply to nested content, e.g. content of nested dictionaries.

---

void **merge**(dictionary: Dictionary, overwrite: bool = false) 🔗

Adds entries from `dictionary` to this dictionary. By default, duplicate keys are not copied over, unless `overwrite` is `true`.

GDScript

```
var dict = { "item": "sword", "quantity": 2 }
var other_dict = { "quantity": 15, "color": "silver" }

# Overwriting of existing keys is disabled by default.
dict.merge(other_dict)
print(dict)  # { "item": "sword", "quantity": 2, "color": "silver" }

# With overwriting of existing keys enabled.
dict.merge(other_dict, true)
print(dict)  # { "item": "sword", "quantity": 15, "color": "silver" }
```

C#

```
var dict = new Godot.Collections.Dictionary
{
    ["item"] = "sword",
    ["quantity"] = 2,
};

var otherDict = new Godot.Collections.Dictionary
{
    ["quantity"] = 15,
    ["color"] = "silver",
};

// Overwriting of existing keys is disabled by default.
dict.Merge(otherDict);
GD.Print(dict); // { "item": "sword", "quantity": 2, "color":
"silver" }

// With overwriting of existing keys enabled.
dict.Merge(otherDict, true);
GD.Print(dict); // { "item": "sword", "quantity": 15, "color":
"silver" }
```

Note: merge is *not* recursive. Nested dictionaries are considered as keys that can be overwritten or not depending on the value of `overwrite`, but they will never be merged together.

Dictionary **merged**(dictionary: Dictionary, overwrite: bool = false) const

Returns a copy of this dictionary merged with the other `dictionary`. By default, duplicate keys are not copied over, unless `overwrite` is `true`. See also merge.

This method is useful for quickly making dictionaries with default values:

```
var base = { "fruit": "apple", "vegetable": "potato" }
var extra = { "fruit": "orange", "dressing": "vinegar" }
# Prints { "fruit": "orange", "vegetable": "potato",
"dressing": "vinegar" }
print(extra.merged(base))
# Prints { "fruit": "apple", "vegetable": "potato", "dressing":
"vinegar" }
print(extra.merged(base, true))
```

---

bool **recursive_equal**(dictionary: Dictionary, recursion_count: int) const 🔗

Returns `true` if the two dictionaries contain the same keys and values, inner **Dictionary** and Array keys and values are compared recursively.

---

int **size**() const 🔗

Returns the number of entries in the dictionary. Empty dictionaries (`{ }`) always return `0`. See also is_empty.

Array **values**() const 🔗

Returns the list of values in this dictionary.

Operator Descriptions

bool **operator !=**(right: Dictionary) 🔗

Returns `true` if the two dictionaries do not contain the same keys and values.

---

bool **operator ==**(right: Dictionary) 🔗

Returns `true` if the two dictionaries contain the same keys and values. The order of the entries does not matter.

Note: In C#, by convention, this operator compares by **reference**. If you need to compare by value, iterate over both dictionaries.

Variant **operator []**(key: Variant) 🔗

Returns the corresponding value for the given `key` in the dictionary. If the entry does not exist, fails and returns `null`. For safe access, use get or has.

float

A built-in type for floating-point numbers.

Description

The **float** built-in type is a 64-bit double-precision floating-point number, equivalent to `double` in C++. This type has 14 reliable decimal digits of precision. The maximum value of **float** is approximately `1.79769e308`, and the minimum is approximately `-1.79769e308`.

Many methods and properties in the engine use 32-bit single-precision floating-point numbers instead, equivalent to `float` in C++, which have 6 reliable decimal digits of precision. For data structures such as Vector2 and Vector3, Godot uses 32-bit floating-point numbers by default, but it can be changed to use 64-bit doubles if Godot is compiled with the `precision=double` option.

Math done using the **float** type is not guaranteed to be exact and will often result in small errors. You should usually use the @GlobalScope.is_equal_approx and @GlobalScope.is_zero_approx methods instead of `==` to compare **float** values for equality.

Tutorials

- Wikipedia: Double-precision floating-point format [https://en.wikipedia.org/wiki/Double-precision_floating-point_format]
- Wikipedia: Single-precision floating-point format [https://en.wikipedia.org/wiki/Single-precision_floating-point_format]

Constructors

float	float()
float	float(from: float)
float	float(from: String)
float	float(from: bool)
float	float(from: int)

Operators

bool	operator !=(right: float)
bool	operator !=(right: int)
Color	operator *(right: Color)
Quaternion	operator *(right: Quaternion)
Vector2	operator *(right: Vector2)

Vector2	operator *(right: Vector2i)
Vector3	operator *(right: Vector3)
Vector3	operator *(right: Vector3i)
Vector4	operator *(right: Vector4)
Vector4	operator *(right: Vector4i)
float	operator *(right: float)
float	operator *(right: int)
float	operator **(right: float)
float	operator **(right: int)
float	operator +(right: float)
float	operator +(right: int)
float	operator -(right: float)
float	operator -(right: int)

float	operator /(right: float)
float	operator /(right: int)
bool	operator <(right: float)
bool	operator <(right: int)
bool	operator <=(right: float)
bool	operator <=(right: int)
bool	operator ==(right: float)
bool	operator ==(right: int)
bool	operator >(right: float)
bool	operator >(right: int)
bool	operator >=(right: float)
bool	operator >=(right: int)
float	operator unary+()

float	operator unary-()

Constructor Descriptions

float **float**()

Constructs a default-initialized **float** set to `0.0`.

float **float**(from: float)

Constructs a **float** as a copy of the given **float**.

float **float**(from: String)

Converts a String to a **float**, following the same rules as String.to_float.

float **float**(from: bool)

Cast a bool value to a floating-point value, `float(true)` will be equal to 1.0 and `float(false)` will be equal to 0.0.

float **float**(from: int)

Cast an int value to a floating-point value, `float(1)` will be equal to `1.0`.

Operator Descriptions

bool **operator !=**(right: float) 🔗

Returns `true` if two floats are different from each other.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator !=**(right: int) 🔗

Returns `true` if the integer has different value than the float.

Color **operator ***(right: Color) 🔗

Multiplies each component of the Color, including the alpha, by the given **float**.

```
print(1.5 * Color(0.5, 0.5, 0.5)) # Prints "(0.75, 0.75, 0.75,
1.5)"
```

Quaternion **operator ***(right: Quaternion) 🔗

Multiplies each component of the Quaternion by the given **float**. This operation is not meaningful on its own, but it can be used as a part of a larger expression.

Vector2 **operator ***(right: Vector2) 🔗

Multiplies each component of the Vector2 by the given **float**.

```
print(2.5 * Vector2(1, 3)) # Prints "(2.5, 7.5)"
```

Vector2 **operator** ***(right: Vector2i)**

Multiplies each component of the Vector2i by the given **float**. Returns a Vector2.

```
print(0.9 * Vector2i(10, 15)) # Prints "(9, 13.5)"
```

Vector3 **operator** ***(right: Vector3)**

Multiplies each component of the Vector3 by the given **float**.

Vector3 **operator** ***(right: Vector3i)**

Multiplies each component of the Vector3i by the given **float**. Returns a Vector3.

```
print(0.9 * Vector3i(10, 15, 20)) # Prints "(9, 13.5, 18)"
```

Vector4 **operator** ***(right: Vector4)**

Multiplies each component of the Vector4 by the given **float**.

Vector4 **operator** ***(right: Vector4i)**

Multiplies each component of the Vector4i by the given **float**. Returns a Vector4.

```
print(0.9 * Vector4i(10, 15, 20, -10)) # Prints "(9, 13.5, 18,
-9)"
```

---

float **operator** ***(right: float)**

Multiplies two **float**s.

---

float operator *(right: int)

Multiplies a **float** and an int. The result is a **float**.

---

float operator **(right: float)

Raises a **float** to a power of a **float**.

```
print(39.0625**0.25) # 2.5
```

---

float operator **(right: int)

Raises a **float** to a power of an int. The result is a **float**.

```
print(0.9**3) # 0.729
```

---

float operator +(right: float)

Adds two floats.

---

float operator +(right: int)

Adds a **float** and an int. The result is a **float**.

---

float operator -(right: float)

Subtracts a float from a float.

---

float operator -(right: int)

Subtracts an int from a **float**. The result is a **float**.

---

float **operator /**(right: float)

Divides two floats.

---

float **operator /**(right: int)

Divides a **float** by an int. The result is a **float**.

---

bool **operator <**(right: float)

Returns `true` if the left float is less than the right one.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator <**(right: int)

Returns `true` if this **float** is less than the given int.

---

bool **operator <=**(right: float)

Returns `true` if the left float is less than or equal to the right one.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator <=**(right: int)

Returns `true` if this **float** is less than or equal to the given int.

---

bool **operator ==**(right: float) 🔗

Returns `true` if both floats are exactly equal.

Note: Due to floating-point precision errors, consider using @GlobalScope.is_equal_approx or @GlobalScope.is_zero_approx instead, which are more reliable.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator ==**(right: int) 🔗

Returns `true` if the **float** and the given int are equal.

---

bool **operator >**(right: float) 🔗

Returns `true` if the left float is greater than the right one.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator >**(right: int) 🔗

Returns `true` if this **float** is greater than the given int.

---

bool **operator >=**(right: float) 🔗

Returns `true` if the left float is greater than or equal to the right one.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator >=**(right: int)

Returns `true` if this **float** is greater than or equal to the given int.

---

float **operator unary+**()

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

---

float **operator unary-**()

Returns the negative value of the **float**. If positive, turns the number negative. If negative, turns the number positive. With floats, the number zero can be either positive or negative.

int

A built-in type for integers.

Description

Signed 64-bit integer type. This means that it can take values from `-2^63` to `2^63 - 1`, i.e. from `-9223372036854775808` to `9223372036854775807`. When it exceeds these bounds, it will wrap around.

ints can be automatically converted to floats when necessary, for example when passing them as arguments in functions. The float will be as close to the original integer as possible.

Likewise, floats can be automatically converted into **int**s. This will truncate the float, discarding anything after the floating-point.

Note: In a boolean context, an **int** will evaluate to `false` if it equals `0`, and to `true` otherwise.

GDScript

```
var x: int = 1 # x is 1
x = 4.2 # x is 4, because 4.2 gets truncated
var max_int = 9223372036854775807 # Biggest value an int can
store
max_int += 1 # max_int is -9223372036854775808, because it
wrapped around
```

C#

```
int x = 1; // x is 1
x = (int)4.2; // x is 4, because 4.2 gets truncated
// We use long below, because GDScript's int is 64-bit while
```

```
C#'s int is 32-bit.
long maxLong = 9223372036854775807; // Biggest value a long can store
maxLong++; // maxLong is now -9223372036854775808, because it wrapped around.

// Alternatively with C#'s 32-bit int type, which has a smaller maximum value.
int maxInt = 2147483647; // Biggest value an int can store
maxInt++; // maxInt is now -2147483648, because it wrapped around
```

You can use the `0b` literal for binary representation, the `0x` literal for hexadecimal representation, and the _ symbol to separate long numbers and improve readability.

GDScript

```
var x = 0b1001  # x is 9
var y = 0xF5 # y is 245
var z = 10_000_000 # z is 10000000
```

C#

```
int x = 0b1001; // x is 9
int y = 0xF5; // y is 245
int z = 10_000_000; // z is 10000000
```

Constructors

int int()

---

int int(from: int)

---

int int(from: String)

int	int(from: bool)
int	int(from: float)

Operators

bool	operator !=(right: float)
bool	operator !=(right: int)
int	operator %(right: int)
int	operator &(right: int)
Color	operator *(right: Color)
Quaternion	operator *(right: Quaternion)
Vector2	operator *(right: Vector2)
Vector2i	operator *(right: Vector2i)
Vector3	operator *(right: Vector3)

Vector3i	operator *(right: Vector3i)
Vector4	operator *(right: Vector4)
Vector4i	operator *(right: Vector4i)
float	operator *(right: float)
int	operator *(right: int)
float	operator **(right: float)
int	operator **(right: int)
float	operator +(right: float)
int	operator +(right: int)
float	operator -(right: float)
int	operator -(right: int)
float	operator /(right: float)
int	operator /(right: int)

bool	operator <(right: float)
bool	operator <(right: int)
int	operator <<(right: int)
bool	operator <=(right: float)
bool	operator <=(right: int)
bool	operator ==(right: float)
bool	operator ==(right: int)
bool	operator >(right: float)
bool	operator >(right: int)
bool	operator >=(right: float)
bool	operator >=(right: int)
int	operator >>(right: int)
int	operator ^(right: int)

int	operator unary+()
int	operator unary-()
int	operator |(right: int)
int	operator ~()

Constructor Descriptions

int **int**() 🔗

Constructs an **int** set to `0`.

---

int **int**(from: int)

Constructs an **int** as a copy of the given **int**.

---

int **int**(from: String)

Constructs a new **int** from a String, following the same rules as String.to_int.

---

int **int**(from: bool)

Constructs a new **int** from a bool. `true` is converted to `1` and `false` is converted to `0`.

---

int **int**(from: float)

Constructs a new **int** from a float. This will truncate the float, discarding anything after the floating point.

Operator Descriptions

bool **operator !=**(right: float)

Returns `true` if the **int** is not equivalent to the float.

---

bool **operator !=**(right: int)

Returns `true` if the **int**s are not equal.

---

int **operator %**(right: int)

Returns the remainder after dividing two **int**s. Uses truncated division, which returns a negative number if the dividend is negative. If this is not desired, consider using @GlobalScope.posmod.

```
print(6 % 2)  # Prints 0
print(11 % 4) # Prints 3
print(-5 % 3) # Prints -2
```

---

int **operator &**(right: int)

Performs the bitwise AND operation.

```
print(0b1100 & 0b1010) # Prints 8 (binary 1000)
```

This is useful for retrieving binary flags from a variable.

```
var flags = 0b101
# Check if the first or second bit are enabled.
if flags & 0b011:
    do_stuff() # This line will run.
```

Color **operator *** (right: Color)

Multiplies each component of the Color by the **int**.

Quaternion **operator *** (right: Quaternion)

Multiplies each component of the Quaternion by the **int**. This operation is not meaningful on its own, but it can be used as a part of a larger expression.

Vector2 **operator *** (right: Vector2)

Multiplies each component of the Vector2 by the **int**.

```
print(2 * Vector2(1, 4)) # Prints (2, 8)
```

Vector2i **operator *** (right: Vector2i)

Multiplies each component of the Vector2i by the **int**.

Vector3 **operator *** (right: Vector3)

Multiplies each component of the Vector3 by the **int**.

Vector3i **operator *** (right: Vector3i)

Multiplies each component of the Vector3i by the **int**.

Vector4 **operator ***(right: Vector4)

Multiplies each component of the Vector4 by the **int**.

Vector4i **operator ***(right: Vector4i)

Multiplies each component of the Vector4i by the **int**.

float **operator ***(right: float)

Multiplies the float by the **int**. The result is a float.

int **operator ***(right: int)

Multiplies the two **int**s.

float **operator ****(right: float)

Raises an **int** to a power of a float. The result is a float.

```
print(2 ** 0.5) # Prints 1.4142135623731
```

int **operator ****(right: int)

Raises the left **int** to a power of the right **int**.

```
print(3 ** 4) # Prints 81
```

float **operator +**(right: float)

Adds the **int** and the float. The result is a float.

---

int **operator +**(right: int)

Adds the two **int**s.

---

float **operator -**(right: float)

Subtracts the float from the **int**. The result is a float.

---

int **operator -**(right: int)

Subtracts the two **int**s.

---

float **operator /**(right: float)

Divides the **int** by the float. The result is a float.

```
print(10 / 3.0) # Prints 3.33333333333333
```

---

int **operator /**(right: int)

Divides the two **int**s. The result is an **int.** This will truncate the float, discarding anything after the floating point.

```
print(6 / 2) # Prints 3
print(5 / 3) # Prints 1
```

---

bool **operator <**(right: float)

Returns `true` if the **int** is less than the float.

---

bool **operator <**(right: int)

Returns `true` if the left **int** is less than the right **int**.

---

int **operator <<**(right: int)

Performs the bitwise shift left operation. Effectively the same as multiplying by a power of 2.

```
print(0b1010 << 1) # Prints 20 (binary 10100)
print(0b1010 << 3) # Prints 80 (binary 1010000)
```

---

bool **operator <=**(right: float)

Returns `true` if the **int** is less than or equal to the float.

---

bool **operator <=**(right: int)

Returns `true` if the left **int** is less than or equal to the right **int**.

---

bool **operator ==**(right: float)

Returns `true` if the **int** is equal to the float.

---

bool **operator ==**(right: int)

Returns `true` if the two **int**s are equal.

---

bool **operator >**(right: float)

Returns `true` if the **int** is greater than the float.

---

bool **operator >**(right: int)

Returns `true` if the left **int** is greater than the right **int**.

bool **operator >=**(right: float)

Returns `true` if the **int** is greater than or equal to the float.

bool **operator >=**(right: int)

Returns `true` if the left **int** is greater than or equal to the right **int**.

int **operator >>**(right: int)

Performs the bitwise shift right operation. Effectively the same as dividing by a power of 2.

```
print(0b1010 >> 1) # Prints 5 (binary 101)
print(0b1010 >> 2) # Prints 2 (binary 10)
```

int **operator ^**(right: int)

Performs the bitwise `XOR` operation.

```
print(0b1100 ^ 0b1010) # Prints 6 (binary 110)
```

int **operator unary+**()

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

int **operator unary-**()

Returns the negated value of the **int**. If positive, turns the number negative. If negative, turns the number positive. If zero, does nothing.

---

int **operator |**(right: int)

Performs the bitwise `OR` operation.

```
print(0b1100 | 0b1010) # Prints 14 (binary 1110)
```

This is useful for storing binary flags in a variable.

```
var flags = 0
flags |= 0b101 # Turn the first and third bits on.
```

---

int **operator ~**()

Performs the bitwise `NOT` operation on the **int**. Due to 2's complement [https://en.wikipedia.org/wiki/Two%27s_complement], it's effectively equal to `-(int + 1)`.

```
print(~4)  # Prints -5
print(~(-7)) # Prints 6
```

NodePath

A pre-parsed scene tree path.

Description

The **NodePath** built-in Variant type represents a path to a node or property in a hierarchy of nodes. It is designed to be efficiently passed into many built-in methods (such as Node.get_node, Object.set_indexed, Tween.tween_property, etc.) without a hard dependence on the node or property they point to.

A node path is represented as a String composed of slash-separated (`/`) node names and colon-separated (`:`) property names (also called "subnames"). Similar to a filesystem path, `".."` and `"."` are special node names. They refer to the parent node and the current node, respectively.

The following examples are paths relative to the current node:

```
^"A"      # Points to the direct child A.
^"A/B"    # Points to A's child B.
^"."      # Points to the current node.
^".."     # Points to the parent node.
^"../C"   # Points to the sibling node C.
^"../.." # Points to the grandparent node.
```

A leading slash means the path is absolute, and begins from the SceneTree:

```
^"/root"           # Points to the SceneTree's root Window.
^"/root/Title"     # May point to the main scene's root node
named "Title".
```

```
^"/root/Global"     # May point to an autoloaded node or scene
named "Global".
```

Despite their name, node paths may also point to a property:

```
^":position"           # Points to this object's position.
^":position:x"         # Points to this object's position in
the x axis.
^"Camera3D:rotation:y" # Points to the child Camera3D and its y
rotation.
^"/root:size:x"        # Points to the root Window and its
width.
```

In some situations, it's possible to omit the leading `:` when pointing to an object's property. As an example, this is the case with Object.set_indexed and Tween.tween_property, as those methods call get_as_property_path under the hood. However, it's generally recommended to keep the `:` prefix.

Node paths cannot check whether they are valid and may point to nodes or properties that do not exist. Their meaning depends entirely on the context in which they're used.

You usually do not have to worry about the **NodePath** type, as strings are automatically converted to the type when necessary. There are still times when defining node paths is useful. For example, exported **NodePath** properties allow you to easily select any node within the currently edited scene. They are also automatically updated when moving, renaming or deleting nodes in the scene tree editor. See also @GDScript.@export_node_path.

See also StringName, which is a similar type designed for optimized strings.

Note: In a boolean context, a **NodePath** will evaluate to `false` if it is empty (`NodePath("")`). Otherwise, a **NodePath** will always evaluate to `true`.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- 2D Role Playing Game (RPG) Demo [https://godotengine.org/asset-library/asset/2729]

Constructors

NodePath	NodePath()
NodePath	NodePath(from: NodePath)
NodePath	NodePath(from: String)

Methods

NodePath	get_as_property_path() const
StringName	get_concatenated_names() const
StringName	get_concatenated_subnames() const

StringName	get_name(idx: int) const
int	get_name_count() const
StringName	get_subname(idx: int) const
int	get_subname_count() const
int	hash() const
bool	is_absolute() const
bool	is_empty() const
NodePath	slice(begin: int, end: int = 2147483647) const

Operators

bool	operator !=(right: NodePath)
bool	operator ==(right: NodePath)

Constructor Descriptions

NodePath **NodePath**()

Constructs an empty **NodePath**.

NodePath **NodePath**(from: NodePath)

Constructs a **NodePath** as a copy of the given **NodePath**.

NodePath **NodePath**(from: String)

Constructs a **NodePath** from a String. The created path is absolute if prefixed with a slash (see is_absolute).

The "subnames" optionally included after the path to the target node can point to properties, and can also be nested.

Examples of strings that could be node paths:

```
# Points to the Sprite2D node.
"Level/RigidBody2D/Sprite2D"

# Points to the Sprite2D node and its "texture" resource.
# get_node() would retrieve the Sprite2D, while
get_node_and_resource()
# would retrieve both the Sprite2D node and the "texture"
resource.
"Level/RigidBody2D/Sprite2D:texture"

# Points to the Sprite2D node and its "position" property.
"Level/RigidBody2D/Sprite2D:position"

# Points to the Sprite2D node and the "x" component of its
"position" property.
"Level/RigidBody2D/Sprite2D:position:x"

# Points to the RigidBody2D node as an absolute path beginning
```

```
from the SceneTree.
"/root/Level/RigidBody2D"
```

Note: In GDScript, it's also possible to convert a constant string into a node path by prefixing it with `^`. `^"path/to/node"` is equivalent to `NodePath("path/to/node")`.

Method Descriptions

NodePath **get_as_property_path**() const

Returns a copy of this node path with a colon character (`:`) prefixed, transforming it to a pure property path with no node names (relative to the current node).

GDScript

```gdscript
# node_path points to the "x" property of the child node named
"position".
var node_path = ^"position:x"

# property_path points to the "position" in the "x" axis of
this node.
var property_path = node_path.get_as_property_path()
print(property_path) # Prints ":position:x"
```

C#

```csharp
// nodePath points to the "x" property of the child node named
"position".
var nodePath = new NodePath("position:x");

// propertyPath points to the "position" in the "x" axis of
this node.
NodePath propertyPath = nodePath.GetAsPropertyPath();
GD.Print(propertyPath); // Prints ":position:x".
```

StringName **get_concatenated_names**() const

Returns all node names concatenated with a slash character (`/`) as a single StringName.

StringName **get_concatenated_subnames**() const

Returns all property subnames concatenated with a colon character (`:`) as a single StringName.

GDScript

```
var node_path = ^"Sprite2D:texture:resource_name"
print(node_path.get_concatenated_subnames()) # Prints "texture:resource_name".
```

C#

```
var nodePath = new NodePath("Sprite2D:texture:resource_name");
GD.Print(nodePath.GetConcatenatedSubnames()); // Prints "texture:resource_name".
```

StringName **get_name**(idx: int) const

Returns the node name indicated by `idx`, starting from 0. If `idx` is out of bounds, an error is generated. See also get_subname_count and get_name_count.

GDScript

```
var sprite_path = NodePath("../RigidBody2D/Sprite2D")
print(sprite_path.get_name(0)) # Prints "..".
print(sprite_path.get_name(1)) # Prints "RigidBody2D".
print(sprite_path.get_name(2)) # Prints "Sprite".
```

C#

```
var spritePath = new NodePath("../RigidBody2D/Sprite2D");
GD.Print(spritePath.GetName(0)); // Prints "..".
GD.Print(spritePath.GetName(1)); // Prints "PathFollow2D".
GD.Print(spritePath.GetName(2)); // Prints "Sprite".
```

int **get_name_count**() const

Returns the number of node names in the path. Property subnames are not included.

For example, `"../RigidBody2D/Sprite2D:texture"` contains 3 node names.

StringName **get_subname**(idx: int) const

Returns the property name indicated by `idx`, starting from 0. If `idx` is out of bounds, an error is generated. See also get_subname_count.

GDScript

```
var path_to_name = NodePath("Sprite2D:texture:resource_name")
print(path_to_name.get_subname(0)) # Prints "texture".
print(path_to_name.get_subname(1)) # Prints "resource_name".
```

C#

```
var pathToName = new
NodePath("Sprite2D:texture:resource_name");
GD.Print(pathToName.GetSubname(0)); // Prints "texture".
GD.Print(pathToName.GetSubname(1)); // Prints "resource_name".
```

int **get_subname_count**() const

Returns the number of property names ("subnames") in the path. Each subname in the node path is listed after a colon character (`:`).

For example, `"Level/RigidBody2D/Sprite2D:texture:resource_name"` contains 2 subnames.

---

int **hash**() const

Returns the 32-bit hash value representing the node path's contents.

Note: Node paths with equal hash values are *not* guaranteed to be the same, as a result of hash collisions. Node paths with different hash values are guaranteed to be different.

---

bool **is_absolute**() const

Returns `true` if the node path is absolute. Unlike a relative path, an absolute path is represented by a leading slash character (`/`) and always begins from the SceneTree. It can be used to reliably access nodes from the root node (e.g. `"/root/Global"` if an autoload named "Global" exists).

---

bool **is_empty**() const

Returns `true` if the node path has been constructed from an empty String (`""`).

NodePath **slice**(begin: int, end: int = 2147483647) const

Returns the slice of the **NodePath**, from `begin` (inclusive) to `end` (exclusive), as a new **NodePath**.

The absolute value of `begin` and `end` will be clamped to the sum of get_name_count and get_subname_count, so the default value for `end` makes it slice to the end of the **NodePath** by default (i.e.

`path.slice(1)` is a shorthand for `path.slice(1, path.get_name_count() + path.get_subname_count())`.

If either `begin` or `end` are negative, they will be relative to the end of the **NodePath** (i.e. `path.slice(0, -2)` is a shorthand for `path.slice(0, path.get_name_count() + path.get_subname_count() - 2)`).

Operator Descriptions

bool **operator !=**(right: NodePath)

Returns `true` if two node paths are not equal.

---

bool **operator ==**(right: NodePath)

Returns `true` if two node paths are equal, that is, they are composed of the same node names and subnames in the same order.

PackedByteArray

A packed array of bytes.

Description

An array specifically designed to hold bytes. Packs data tightly, so it saves memory for large array sizes.

PackedByteArray also provides methods to encode/decode various types to/from bytes. The way values are encoded is an implementation detail and shouldn't be relied upon when interacting with external apps.

Note: Packed arrays are always passed by reference. To get a copy of an array that can be modified independently of the original array, use duplicate. This is *not* the case for built-in properties and methods. The returned packed array of these are a copies, and changing it will *not* affect the original value. To update a built-in property you need to modify the returned array, and then assign it to the property again.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

PackedByteArray PackedByteArray()

PackedByteArray	PackedByteArray(from: PackedByteArray)
PackedByteArray	PackedByteArray(from: Array)

Methods

bool	append(value: int)
void	append_array(array: PackedByteArray)
int	bsearch(value: int, before: bool = true)
void	clear()
PackedByteArray	compress(compression_mode: int = 0) const
int	count(value: int) const
float	decode_double(byte_offset: int) const
float	decode_float(byte_offset: int) const
float	decode_half(byte_offset: int) const

int	decode_s8(byte_offset: int) const
int	decode_s16(byte_offset: int) const
int	decode_s32(byte_offset: int) const
int	decode_s64(byte_offset: int) const
int	decode_u8(byte_offset: int) const
int	decode_u16(byte_offset: int) const
int	decode_u32(byte_offset: int) const
int	decode_u64(byte_offset: int) const
Variant	decode_var(byte_offset: int, allow_objects: bool = false) const
int	decode_var_size(byte_offset: int, allow_objects: bool = false) const
PackedByteArray	decompress(buffer_size: int, compression_mode: int = 0) const

PackedByteArray	decompress_dynamic(max_output_size: int, compression_mode: int = 0) const
PackedByteArray	duplicate()
void	encode_double(byte_offset: int, value: float)
void	encode_float(byte_offset: int, value: float)
void	encode_half(byte_offset: int, value: float)
void	encode_s8(byte_offset: int, value: int)
void	encode_s16(byte_offset: int, value: int)
void	encode_s32(byte_offset: int, value: int)
void	encode_s64(byte_offset: int, value: int)
void	encode_u8(byte_offset: int, value: int)
void	encode_u16(byte_offset: int, value: int)
void	encode_u32(byte_offset: int, value: int)

void	encode_u64(byte_offset: int, value: int)
int	encode_var(byte_offset: int, value: Variant, allow_objects: bool = false)
void	fill(value: int)
int	find(value: int, from: int = 0) const
String	get_string_from_ascii() const
String	get_string_from_utf8() const
String	get_string_from_utf16() const
String	get_string_from_utf32() const
String	get_string_from_wchar() const
bool	has(value: int) const
bool	has_encoded_var(byte_offset: int, allow_objects: bool = false) const
String	hex_encode() const

int	insert(at_index: int, value: int)
bool	is_empty() const
bool	push_back(value: int)
void	remove_at(index: int)
int	resize(new_size: int)
void	reverse()
int	rfind(value: int, from: int = -1) const
void	set(index: int, value: int)
int	size() const
PackedByteArray	slice(begin: int, end: int = 2147483647) const
void	sort()
PackedFloat32Array	to_float32_array() const
PackedFloat64Array	to_float64_array() const

PackedInt32Array	to_int32_array() const
PackedInt64Array	to_int64_array() const

Operators

bool	operator !=(right: PackedByteArray)
PackedByteArray	operator +(right: PackedByteArray)
bool	operator ==(right: PackedByteArray)
int	operator [](index: int)

Constructor Descriptions

PackedByteArray **PackedByteArray**() 🔗

Constructs an empty **PackedByteArray**.

PackedByteArray **PackedByteArray**(from: PackedByteArray)

Constructs a **PackedByteArray** as a copy of the given **PackedByteArray**.

PackedByteArray **PackedByteArray**(from: Array)

Constructs a new **PackedByteArray**. Optionally, you can pass in a generic Array that will be converted.

Method Descriptions

bool **append**(value: int)

Appends an element at the end of the array (alias of push_back).

---

void **append_array**(array: PackedByteArray)

Appends a **PackedByteArray** at the end of this array.

---

int **bsearch**(value: int, before: bool = true)

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a `before` specifier can be passed. If `false`, the returned index comes after all existing entries of the value in the array.

Note: Calling bsearch on an unsorted array results in unexpected behavior.

---

void **clear**()

Clears the array. This is equivalent to using resize with a size of `0`.

PackedByteArray **compress**(compression_mode: int = 0) const

Returns a new **PackedByteArray** with the data compressed. Set the compression mode using one of CompressionMode's constants.

---

int **count**(value: int) const

Returns the number of times an element is in the array.

---

float **decode_double**(byte_offset: int) const

Decodes a 64-bit floating-point number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0.0` if a valid number can't be decoded.

---

float **decode_float**(byte_offset: int) const

Decodes a 32-bit floating-point number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0.0` if a valid number can't be decoded.

---

float **decode_half**(byte_offset: int) const

Decodes a 16-bit floating-point number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0.0` if a valid number can't be decoded.

---

int **decode_s8**(byte_offset: int) const

Decodes a 8-bit signed integer number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0` if a valid number can't be decoded.

---

int **decode_s16**(byte_offset: int) const 🔗

Decodes a 16-bit signed integer number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0` if a valid number can't be decoded.

int **decode_s32**(byte_offset: int) const 🔗

Decodes a 32-bit signed integer number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0` if a valid number can't be decoded.

int **decode_s64**(byte_offset: int) const 🔗

Decodes a 64-bit signed integer number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0` if a valid number can't be decoded.

int **decode_u8**(byte_offset: int) const 🔗

Decodes a 8-bit unsigned integer number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0` if a valid number can't be decoded.

int **decode_u16**(byte_offset: int) const 🔗

Decodes a 16-bit unsigned integer number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0` if a valid number can't be decoded.

int **decode_u32**(byte_offset: int) const 🔗

Decodes a 32-bit unsigned integer number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0` if a valid number can't be decoded.

---

int **decode_u64**(byte_offset: int) const 🔗

Decodes a 64-bit unsigned integer number from the bytes starting at `byte_offset`. Fails if the byte count is insufficient. Returns `0` if a valid number can't be decoded.

Variant **decode_var**(byte_offset: int, allow_objects: bool = false) const 🔗

Decodes a Variant from the bytes starting at `byte_offset`. Returns `null` if a valid variant can't be decoded or the value is Object-derived and `allow_objects` is `false`.

---

int **decode_var_size**(byte_offset: int, allow_objects: bool = false) const 🔗

Decodes a size of a Variant from the bytes starting at `byte_offset`. Requires at least 4 bytes of data starting at the offset, otherwise fails.

PackedByteArray **decompress**(buffer_size: int, compression_mode: int = 0) const 🔗

Returns a new **PackedByteArray** with the data decompressed. Set `buffer_size` to the size of the uncompressed data. Set the compression mode using one of CompressionMode's constants.

Note: Decompression is not guaranteed to work with data not compressed by Godot, for example if data compressed with the deflate compression mode lacks a checksum or header.

PackedByteArray **decompress_dynamic**(max_output_size: int, compression_mode: int = 0) const 🔗

Returns a new **PackedByteArray** with the data decompressed. Set the compression mode using one of CompressionMode's constants. **This method only accepts brotli, gzip, and deflate compression modes.**

This method is potentially slower than decompress, as it may have to re-allocate its output buffer multiple times while decompressing, whereas decompress knows it's output buffer size from the beginning.

GZIP has a maximal compression ratio of 1032:1, meaning it's very possible for a small compressed payload to decompress to a potentially very large output. To guard against this, you may provide a maximum size this function is allowed to allocate in bytes via `max_output_size`. Passing -1 will allow for unbounded output. If any positive value is passed, and the decompression exceeds that amount in bytes, then an error will be returned.

Note: Decompression is not guaranteed to work with data not compressed by Godot, for example if data compressed with the deflate compression mode lacks a checksum or header.

PackedByteArray **duplicate**() 🔗

Creates a copy of the array, and returns it.

---

void **encode_double**(byte_offset: int, value: float)

Encodes a 64-bit floating-point number as bytes at the index of `byte_offset` bytes. The array must have at least 8 bytes of allocated space, starting at the offset.

---

void **encode_float**(byte_offset: int, value: float)

Encodes a 32-bit floating-point number as bytes at the index of `byte_offset` bytes. The array must have at least 4 bytes of space, starting at the offset.

---

void **encode_half**(byte_offset: int, value: float)

Encodes a 16-bit floating-point number as bytes at the index of `byte_offset` bytes. The array must have at least 2 bytes of space, starting at the offset.

---

void **encode_s8**(byte_offset: int, value: int)

Encodes a 8-bit signed integer number (signed byte) at the index of `byte_offset` bytes. The array must have at least 1 byte of space, starting at the offset.

---

void **encode_s16**(byte_offset: int, value: int)

Encodes a 16-bit signed integer number as bytes at the index of `byte_offset` bytes. The array must have at least 2 bytes of space, starting at the offset.

---

void **encode_s32**(byte_offset: int, value: int)

Encodes a 32-bit signed integer number as bytes at the index of `byte_offset` bytes. The array must have at least 4 bytes of space, starting at the offset.

---

void **encode_s64**(byte_offset: int, value: int)

Encodes a 64-bit signed integer number as bytes at the index of `byte_offset` bytes. The array must have at least 8 bytes of space, starting at the offset.

---

void **encode_u8**(byte_offset: int, value: int)

Encodes a 8-bit unsigned integer number (byte) at the index of `byte_offset` bytes. The array must have at least 1 byte of space, starting at the offset.

---

void **encode_u16**(byte_offset: int, value: int)

Encodes a 16-bit unsigned integer number as bytes at the index of `byte_offset` bytes. The array must have at least 2 bytes of space, starting at the offset.

---

void **encode_u32**(byte_offset: int, value: int)

Encodes a 32-bit unsigned integer number as bytes at the index of `byte_offset` bytes. The array must have at least 4 bytes of space, starting at the offset.

---

void **encode_u64**(byte_offset: int, value: int)

Encodes a 64-bit unsigned integer number as bytes at the index of `byte_offset` bytes. The array must have at least 8 bytes of space, starting at the offset.

---

int **encode_var**(byte_offset: int, value: Variant, allow_objects: bool = false)

Encodes a Variant at the index of `byte_offset` bytes. A sufficient space must be allocated, depending on the encoded variant's size. If `allow_objects` is `false`, Object-derived values are not permitted and will instead be serialized as ID-only.

---

void **fill**(value: int)

Assigns the given value to all elements in the array. This can typically be used together with resize to create an array with a given size and initialized elements.

---

int **find**(value: int, from: int = 0) const

Searches the array for a value and returns its index or `-1` if not found. Optionally, the initial search index can be passed.

String **get_string_from_ascii**() const

Converts ASCII/Latin-1 encoded array to String. Fast alternative to get_string_from_utf8 if the content is ASCII/Latin-1 only. Unlike the UTF-8 function this function maps every byte to a character in the array. Multibyte sequences will not be interpreted correctly. For parsing user input always use get_string_from_utf8. This is the inverse of String.to_ascii_buffer.

String **get_string_from_utf8**() const 🔗

Converts UTF-8 encoded array to String. Slower than get_string_from_ascii but supports UTF-8 encoded data. Use this function if you are unsure about the source of the data. For user input this function should always be preferred. Returns empty string if source array is not valid UTF-8 string. This is the inverse of String.to_utf8_buffer.

String **get_string_from_utf16**() const 🔗

Converts UTF-16 encoded array to String. If the BOM is missing, system endianness is assumed. Returns empty string if source array is not valid UTF-16 string. This is the inverse of String.to_utf16_buffer.

String **get_string_from_utf32**() const 🔗

Converts UTF-32 encoded array to String. System endianness is assumed. Returns empty string if source array is not valid UTF-32 string. This is the inverse of String.to_utf32_buffer.

String **get_string_from_wchar**() const 🔗

Converts wide character (`wchar_t`, UTF-16 on Windows, UTF-32 on other platforms) encoded array to String. Returns empty string if source array is not valid wide string. This is the inverse of String.to_wchar_buffer.

---

bool **has**(value: int) const 🔗

Returns `true` if the array contains `value`.

bool **has_encoded_var**(byte_offset: int, allow_objects: bool = false) const

Returns `true` if a valid Variant value can be decoded at the `byte_offset`. Returns `false` otherwise or when the value is Object-derived and `allow_objects` is `false`.

String **hex_encode**() const

Returns a hexadecimal representation of this array as a String.

GDScript

```
var array = PackedByteArray([11, 46, 255])
print(array.hex_encode()) # Prints: 0b2eff
```

C#

```
var array = new byte[] {11, 46, 255};
GD.Print(array.HexEncode()); // Prints: 0b2eff
```

int **insert**(at_index: int, value: int)

Inserts a new element at a given position in the array. The position must be valid, or at the end of the array (`idx == size()`).

bool **is_empty**() const

Returns `true` if the array is empty.

bool **push_back**(value: int)

Appends an element at the end of the array.

---

void **remove_at**(index: int)

Removes an element from the array by index.

---

int **resize**(new_size: int)

Sets the size of the array. If the array is grown, reserves elements at the end of the array. If the array is shrunk, truncates the array to the new size. Calling resize once and assigning the new values is faster than adding new elements one by one.

---

void **reverse**()

Reverses the order of the elements in the array.

---

int **rfind**(value: int, from: int = -1) const

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

---

void **set**(index: int, value: int)

Changes the byte at the given index.

---

int **size**() const

Returns the number of elements in the array.

PackedByteArray **slice**(begin: int, end: int = 2147483647) const

Returns the slice of the **PackedByteArray**, from `begin` (inclusive) to `end` (exclusive), as a new **PackedByteArray**.

The absolute value of `begin` and `end` will be clamped to the array size, so the default value for `end` makes it slice to the size of the array by default (i.e. `arr.slice(1)` is a shorthand for `arr.slice(1, arr.size())`).

If either `begin` or `end` are negative, they will be relative to the end of the array (i.e. `arr.slice(0, -2)` is a shorthand for `arr.slice(0, arr.size() - 2)`).

---

void **sort**() 🔗

Sorts the elements of the array in ascending order.

PackedFloat32Array **to_float32_array**() const 🔗

Returns a copy of the data converted to a PackedFloat32Array, where each block of 4 bytes has been converted to a 32-bit float (C++ `float`).

The size of the input array must be a multiple of 4 (size of 32-bit float). The size of the new array will be `byte_array.size() / 4`.

If the original data can't be converted to 32-bit floats, the resulting data is undefined.

PackedFloat64Array **to_float64_array**() const 🔗

Returns a copy of the data converted to a PackedFloat64Array, where each block of 8 bytes has been converted to a 64-bit float (C++ `double`, Godot float).

The size of the input array must be a multiple of 8 (size of 64-bit double). The size of the new array will be `byte_array.size() / 8`.

If the original data can't be converted to 64-bit floats, the resulting data is undefined.

PackedInt32Array **to_int32_array**() const

Returns a copy of the data converted to a PackedInt32Array, where each block of 4 bytes has been converted to a signed 32-bit integer (C++ `int32_t`).

The size of the input array must be a multiple of 4 (size of 32-bit integer). The size of the new array will be `byte_array.size() / 4`.

If the original data can't be converted to signed 32-bit integers, the resulting data is undefined.

PackedInt64Array **to_int64_array**() const

Returns a copy of the data converted to a PackedInt64Array, where each block of 8 bytes has been converted to a signed 64-bit integer (C++ `int64_t`, Godot int).

The size of the input array must be a multiple of 8 (size of 64-bit integer). The size of the new array will be `byte_array.size() / 8`.

If the original data can't be converted to signed 64-bit integers, the resulting data is undefined.

Operator Descriptions

bool **operator !=**(right: PackedByteArray)

Returns `true` if contents of the arrays differ.

PackedByteArray **operator +**(right: PackedByteArray)

Returns a new **PackedByteArray** with contents of `right` added at the end of this array. For better performance, consider using append_array instead.

---

bool **operator ==**(right: PackedByteArray)

Returns `true` if contents of both arrays are the same, i.e. they have all equal bytes at the corresponding indices.

---

int **operator []**(index: int)

Returns the byte at index `index`. Negative indices can be used to access the elements starting from the end. Using index out of array's bounds will result in an error.

Note that the byte is returned as a 64-bit int.

PackedColorArray

A packed array of Colors.

Description

An array specifically designed to hold Color. Packs data tightly, so it saves memory for large array sizes.

Differences between packed arrays, typed arrays, and untyped arrays: Packed arrays are generally faster to iterate on and modify compared to a typed array of the same type (e.g. **PackedColorArray** versus `Array[Color]`). Also, packed arrays consume less memory. As a downside, packed arrays are less flexible as they don't offer as many convenience methods such as Array.map. Typed arrays are in turn faster to iterate on and modify than untyped arrays.

Note: Packed arrays are always passed by reference. To get a copy of an array that can be modified independently of the original array, use duplicate. This is *not* the case for built-in properties and methods. The returned packed array of these are a copies, and changing it will *not* affect the original value. To update a built-in property you need to modify the returned array, and then assign it to the property again.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

PackedColorArray	PackedColorArray()
PackedColorArray	PackedColorArray(from: PackedColorArray)
PackedColorArray	PackedColorArray(from: Array)

Methods

bool	append(value: Color)
void	append_array(array: PackedColorArray)
int	bsearch(value: Color, before: bool = true)
void	clear()
int	count(value: Color) const
PackedColorArray	duplicate()
void	fill(value: Color)

int	find(value: Color, from: int = 0) const
bool	has(value: Color) const
int	insert(at_index: int, value: Color)
bool	is_empty() const
bool	push_back(value: Color)
void	remove_at(index: int)
int	resize(new_size: int)
void	reverse()
int	rfind(value: Color, from: int = -1) const
void	set(index: int, value: Color)
int	size() const
PackedColorArray	slice(begin: int, end: int = 2147483647) const
void	sort()

PackedByteArray	to_byte_array() const

Operators

bool	operator !=(right: PackedColorArray)
PackedColorArray	operator +(right: PackedColorArray)
bool	operator ==(right: PackedColorArray)
Color	operator [](index: int)

Constructor Descriptions

PackedColorArray **PackedColorArray**() 🔗

Constructs an empty **PackedColorArray**.

PackedColorArray **PackedColorArray**(from: PackedColorArray)

Constructs a **PackedColorArray** as a copy of the given **PackedColorArray**.

PackedColorArray **PackedColorArray**(from: Array)

Constructs a new **PackedColorArray**. Optionally, you can pass in a generic Array that will be converted.

Note: When initializing a **PackedColorArray** with elements, it must be initialized with an Array of Color values:

```
var array = PackedColorArray([Color(0.1, 0.2, 0.3), Color(0.4, 0.5, 0.6)])
```

Method Descriptions

bool **append**(value: Color)

Appends an element at the end of the array (alias of push_back).

---

void **append_array**(array: PackedColorArray)

Appends a **PackedColorArray** at the end of this array.

---

int **bsearch**(value: Color, before: bool = true)

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a `before` specifier can be passed. If `false`, the returned index comes after all existing entries of the value in the array.

Note: Calling bsearch on an unsorted array results in unexpected behavior.

---

void **clear**()

Clears the array. This is equivalent to using resize with a size of `0`.

---

int **count**(value: Color) const 🔗

Returns the number of times an element is in the array.

PackedColorArray **duplicate**() 🔗

Creates a copy of the array, and returns it.

---

void **fill**(value: Color) 🔗

Assigns the given value to all elements in the array. This can typically be used together with resize to create an array with a given size and initialized elements.

---

int **find**(value: Color, from: int = 0) const 🔗

Searches the array for a value and returns its index or `-1` if not found. Optionally, the initial search index can be passed.

---

bool **has**(value: Color) const 🔗

Returns `true` if the array contains `value`.

---

int **insert**(at_index: int, value: Color) 🔗

Inserts a new element at a given position in the array. The position must be valid, or at the end of the array (`idx == size()`).

---

bool **is_empty**() const 🔗

Returns `true` if the array is empty.

---

bool **push_back**(value: Color) 🔗

Appends a value to the array.

---

void **remove_at**(index: int) 🔗

Removes an element from the array by index.

---

int **resize**(new_size: int) 🔗

Sets the size of the array. If the array is grown, reserves elements at the end of the array. If the array is shrunk, truncates the array to the new size. Calling resize once and assigning the new values is faster than adding new elements one by one.

---

void **reverse**() 🔗

Reverses the order of the elements in the array.

---

int **rfind**(value: Color, from: int = -1) const 🔗

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

---

void **set**(index: int, value: Color) 🔗

Changes the Color at the given index.

---

int **size**() const

Returns the number of elements in the array.

PackedColorArray **slice**(begin: int, end: int = 2147483647) const

Returns the slice of the **PackedColorArray**, from `begin` (inclusive) to `end` (exclusive), as a new **PackedColorArray**.

The absolute value of `begin` and `end` will be clamped to the array size, so the default value for `end` makes it slice to the size of the array by default (i.e. `arr.slice(1)` is a shorthand for `arr.slice(1, arr.size())`).

If either `begin` or `end` are negative, they will be relative to the end of the array (i.e. `arr.slice(0, -2)` is a shorthand for `arr.slice(0, arr.size() - 2)`).

---

void **sort**()

Sorts the elements of the array in ascending order.

PackedByteArray **to_byte_array**() const

Returns a PackedByteArray with each color encoded as bytes.

Operator Descriptions

bool **operator !=**(right: PackedColorArray)

Returns `true` if contents of the arrays differ.

PackedColorArray **operator +**(right: PackedColorArray)

Returns a new **PackedColorArray** with contents of `right` added at the end of this array. For better performance, consider using append_array instead.

---

bool **operator ==**(right: PackedColorArray)

Returns `true` if contents of both arrays are the same, i.e. they have all equal Colors at the corresponding indices.

Color **operator []**(index: int)

Returns the Color at index `index`. Negative indices can be used to access the elements starting from the end. Using index out of array's bounds will result in an error.

PackedFloat32Array

A packed array of 32-bit floating-point values.

Description

An array specifically designed to hold 32-bit floating-point values (float). Packs data tightly, so it saves memory for large array sizes.

If you need to pack 64-bit floats tightly, see PackedFloat64Array.

Note: Packed arrays are always passed by reference. To get a copy of an array that can be modified independently of the original array, use duplicate. This is *not* the case for built-in properties and methods. The returned packed array of these are a copies, and changing it will *not* affect the original value. To update a built-in property you need to modify the returned array, and then assign it to the property again.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

PackedFloat32Array PackedFloat32Array()

PackedFloat32Array	PackedFloat32Array(from: PackedFloat32Array)
PackedFloat32Array	PackedFloat32Array(from: Array)

Methods

bool	append(value: float)
void	append_array(array: PackedFloat32Array)
int	bsearch(value: float, before: bool = true)
void	clear()
int	count(value: float) const
PackedFloat32Array	duplicate()
void	fill(value: float)
int	find(value: float, from: int = 0) const
bool	has(value: float) const

int	insert(at_index: int, value: float)
bool	is_empty() const
bool	push_back(value: float)
void	remove_at(index: int)
int	resize(new_size: int)
void	reverse()
int	rfind(value: float, from: int = -1) const
void	set(index: int, value: float)
int	size() const
PackedFloat32Array	slice(begin: int, end: int = 2147483647) const
void	sort()
PackedByteArray	to_byte_array() const

Operators

bool	operator !=(right: PackedFloat32Array)
PackedFloat32Array	operator +(right: PackedFloat32Array)
bool	operator ==(right: PackedFloat32Array)
float	operator [](index: int)

Constructor Descriptions

PackedFloat32Array **PackedFloat32Array**() 🔗

Constructs an empty **PackedFloat32Array**.

PackedFloat32Array **PackedFloat32Array**(from: PackedFloat32Array)

Constructs a **PackedFloat32Array** as a copy of the given **PackedFloat32Array**.

PackedFloat32Array **PackedFloat32Array**(from: Array)

Constructs a new **PackedFloat32Array**. Optionally, you can pass in a generic Array that will be converted.

Method Descriptions

bool **append**(value: float) 🔗

Appends an element at the end of the array (alias of push_back).

---

void **append_array**(array: PackedFloat32Array) 🔗

Appends a **PackedFloat32Array** at the end of this array.

---

int **bsearch**(value: float, before: bool = true) 🔗

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a `before` specifier can be passed. If `false`, the returned index comes after all existing entries of the value in the array.

Note: Calling bsearch on an unsorted array results in unexpected behavior.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

---

void **clear**() 🔗

Clears the array. This is equivalent to using resize with a size of `0`.

---

int **count**(value: float) const 🔗

Returns the number of times an element is in the array.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

PackedFloat32Array **duplicate**()

Creates a copy of the array, and returns it.

---

void **fill**(value: float)

Assigns the given value to all elements in the array. This can typically be used together with resize to create an array with a given size and initialized elements.

---

int **find**(value: float, from: int = 0) const

Searches the array for a value and returns its index or `-1` if not found. Optionally, the initial search index can be passed.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

---

bool **has**(value: float) const

Returns `true` if the array contains `value`.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

---

int **insert**(at_index: int, value: float)

Inserts a new element at a given position in the array. The position must be valid, or at the end of the array (`idx == size()`).

---

bool **is_empty**() const

Returns `true` if the array is empty.

---

bool **push_back**(value: float)

Appends an element at the end of the array.

---

void **remove_at**(index: int)

Removes an element from the array by index.

---

int **resize**(new_size: int)

Sets the size of the array. If the array is grown, reserves elements at the end of the array. If the array is shrunk, truncates the array to the new size. Calling resize once and assigning the new values is faster than adding new elements one by one.

---

void **reverse**()

Reverses the order of the elements in the array.

---

int **rfind**(value: float, from: int = -1) const

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

void **set**(index: int, value: float)

Changes the float at the given index.

int **size**() const

Returns the number of elements in the array.

PackedFloat32Array **slice**(begin: int, end: int = 2147483647) const

Returns the slice of the **PackedFloat32Array**, from `begin` (inclusive) to `end` (exclusive), as a new **PackedFloat32Array**.

The absolute value of `begin` and `end` will be clamped to the array size, so the default value for `end` makes it slice to the size of the array by default (i.e. `arr.slice(1)` is a shorthand for `arr.slice(1, arr.size())`).

If either `begin` or `end` are negative, they will be relative to the end of the array (i.e. `arr.slice(0, -2)` is a shorthand for `arr.slice(0, arr.size() - 2)`).

void **sort**()

Sorts the elements of the array in ascending order.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

PackedByteArray **to_byte_array**() const 🔗

Returns a copy of the data converted to a PackedByteArray, where each element have been encoded as 4 bytes.

The size of the new array will be `float32_array.size() * 4`.

Operator Descriptions

bool **operator !=**(right: PackedFloat32Array) 🔗

Returns `true` if contents of the arrays differ.

PackedFloat32Array **operator +**(right: PackedFloat32Array) 🔗

Returns a new **PackedFloat32Array** with contents of `right` added at the end of this array. For better performance, consider using append_array instead.

---

bool **operator ==**(right: PackedFloat32Array) 🔗

Returns `true` if contents of both arrays are the same, i.e. they have all equal floats at the corresponding indices.

---

float **operator []**(index: int) 🔗

Returns the float at index `index`. Negative indices can be used to access the elements starting from the end. Using index out of array's bounds will result in an error.

Note that float type is 64-bit, unlike the values stored in the array.

PackedFloat64Array

A packed array of 64-bit floating-point values.

Description

An array specifically designed to hold 64-bit floating-point values (double). Packs data tightly, so it saves memory for large array sizes.

If you only need to pack 32-bit floats tightly, see PackedFloat32Array for a more memory-friendly alternative.

Differences between packed arrays, typed arrays, and untyped arrays: Packed arrays are generally faster to iterate on and modify compared to a typed array of the same type (e.g. **PackedFloat64Array** versus `Array[float]`). Also, packed arrays consume less memory. As a downside, packed arrays are less flexible as they don't offer as many convenience methods such as Array.map. Typed arrays are in turn faster to iterate on and modify than untyped arrays.

Note: Packed arrays are always passed by reference. To get a copy of an array that can be modified independently of the original array, use duplicate. This is *not* the case for built-in properties and methods. The returned packed array of these are a copies, and changing it will *not* affect the original value. To update a built-in property you need to modify the returned array, and then assign it to the property again.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

PackedFloat64Array	PackedFloat64Array()
PackedFloat64Array	PackedFloat64Array(from: PackedFloat64Array)
PackedFloat64Array	PackedFloat64Array(from: Array)

Methods

bool	append(value: float)
void	append_array(array: PackedFloat64Array)
int	bsearch(value: float, before: bool = true)
void	clear()
int	count(value: float) const

PackedFloat64Array	duplicate()
void	fill(value: float)
int	find(value: float, from: int = 0) const
bool	has(value: float) const
int	insert(at_index: int, value: float)
bool	is_empty() const
bool	push_back(value: float)
void	remove_at(index: int)
int	resize(new_size: int)
void	reverse()
int	rfind(value: float, from: int = -1) const
void	set(index: int, value: float)
int	size() const

PackedFloat64Array	slice(begin: int, end: int = 2147483647) const
void	sort()
PackedByteArray	to_byte_array() const

Operators

bool	operator !=(right: PackedFloat64Array)
PackedFloat64Array	operator +(right: PackedFloat64Array)
bool	operator ==(right: PackedFloat64Array)
float	operator [](index: int)

Constructor Descriptions

PackedFloat64Array **PackedFloat64Array**()

Constructs an empty **PackedFloat64Array**.

PackedFloat64Array **PackedFloat64Array**(from: PackedFloat64Array)

Constructs a **PackedFloat64Array** as a copy of the given **PackedFloat64Array**.

PackedFloat64Array **PackedFloat64Array**(from: Array)

Constructs a new **PackedFloat64Array**. Optionally, you can pass in a generic Array that will be converted.

Method Descriptions

bool **append**(value: float)

Appends an element at the end of the array (alias of push_back).

void **append_array**(array: PackedFloat64Array)

Appends a **PackedFloat64Array** at the end of this array.

int **bsearch**(value: float, before: bool = true)

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a `before` specifier can be passed. If `false`, the returned index comes after all existing entries of the value in the array.

Note: Calling bsearch on an unsorted array results in unexpected behavior.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

void **clear**()

Clears the array. This is equivalent to using resize with a size of `0`.

int **count**(value: float) const

Returns the number of times an element is in the array.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

PackedFloat64Array **duplicate**()

Creates a copy of the array, and returns it.

void **fill**(value: float)

Assigns the given value to all elements in the array. This can typically be used together with resize to create an array with a given size and initialized elements.

int **find**(value: float, from: int = 0) const

Searches the array for a value and returns its index or `-1` if not found. Optionally, the initial search index can be passed.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

bool **has**(value: float) const 🔗

Returns `true` if the array contains `value`.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

---

int **insert**(at_index: int, value: float) 🔗

Inserts a new element at a given position in the array. The position must be valid, or at the end of the array (`idx == size()`).

---

bool **is_empty**() const 🔗

Returns `true` if the array is empty.

---

bool **push_back**(value: float) 🔗

Appends an element at the end of the array.

---

void **remove_at**(index: int) 🔗

Removes an element from the array by index.

---

int **resize**(new_size: int) 🔗

Sets the size of the array. If the array is grown, reserves elements at the end of the array. If the array is shrunk, truncates the array to the new size. Calling resize once and assigning the new values is faster than adding new elements one by one.

---

void **reverse**()

Reverses the order of the elements in the array.

---

int **rfind**(value: float, from: int = -1) const

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

---

void **set**(index: int, value: float)

Changes the float at the given index.

---

int **size**() const

Returns the number of elements in the array.

PackedFloat64Array **slice**(begin: int, end: int = 2147483647) const

Returns the slice of the **PackedFloat64Array**, from `begin` (inclusive) to `end` (exclusive), as a new **PackedFloat64Array**.

The absolute value of `begin` and `end` will be clamped to the array size, so the default value for `end` makes it slice to the size of the array by default (i.e. `arr.slice(1)` is a shorthand for `arr.slice(1, arr.size())`).

If either `begin` or `end` are negative, they will be relative to the end of the array (i.e. `arr.slice(0, -2)` is a shorthand for `arr.slice(0, arr.size() - 2)`).

---

void **sort**()

Sorts the elements of the array in ascending order.

Note: @GDScript.NAN doesn't behave the same as other numbers. Therefore, the results from this method may not be accurate if NaNs are included.

PackedByteArray **to_byte_array**() const

Returns a copy of the data converted to a PackedByteArray, where each element have been encoded as 8 bytes.

The size of the new array will be `float64_array.size() * 8`.

Operator Descriptions

bool **operator !=**(right: PackedFloat64Array)

Returns `true` if contents of the arrays differ.

PackedFloat64Array **operator +**(right: PackedFloat64Array)

Returns a new **PackedFloat64Array** with contents of `right` added at the end of this array. For better performance, consider using append_array instead.

---

bool **operator ==**(right: PackedFloat64Array)

Returns `true` if contents of both arrays are the same, i.e. they have all equal doubles at the corresponding indices.

---

float **operator []**(index: int)

Returns the float at index `index`. Negative indices can be used to access the elements starting from the end. Using index out of array's bounds will result in an error.

PackedInt32Array

A packed array of 32-bit integers.

Description

An array specifically designed to hold 32-bit integer values. Packs data tightly, so it saves memory for large array sizes.

Note: This type stores signed 32-bit integers, which means it can take values in the interval `[-2^31, 2^31 - 1]`, i.e. `[-2147483648, 2147483647]`. Exceeding those bounds will wrap around. In comparison, int uses signed 64-bit integers which can hold much larger values. If you need to pack 64-bit integers tightly, see PackedInt64Array.

Note: Packed arrays are always passed by reference. To get a copy of an array that can be modified independently of the original array, use duplicate. This is *not* the case for built-in properties and methods. The returned packed array of these are a copies, and changing it will *not* affect the original value. To update a built-in property you need to modify the returned array, and then assign it to the property again.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

PackedInt32Array	PackedInt32Array()
PackedInt32Array	PackedInt32Array(from: PackedInt32Array)
PackedInt32Array	PackedInt32Array(from: Array)

Methods

bool	append(value: int)
void	append_array(array: PackedInt32Array)
int	bsearch(value: int, before: bool = true)
void	clear()
int	count(value: int) const
PackedInt32Array	duplicate()
void	fill(value: int)
int	find(value: int, from: int = 0) const

bool	has(value: int) const
int	insert(at_index: int, value: int)
bool	is_empty() const
bool	push_back(value: int)
void	remove_at(index: int)
int	resize(new_size: int)
void	reverse()
int	rfind(value: int, from: int = -1) const
void	set(index: int, value: int)
int	size() const
PackedInt32Array	slice(begin: int, end: int = 2147483647) const
void	sort()
PackedByteArray	to_byte_array() const

Operators

bool	operator !=(right: PackedInt32Array)
PackedInt32Array	operator +(right: PackedInt32Array)
bool	operator ==(right: PackedInt32Array)
int	operator [](index: int)

Constructor Descriptions

PackedInt32Array **PackedInt32Array**() 🔗

Constructs an empty **PackedInt32Array**.

PackedInt32Array **PackedInt32Array**(from: PackedInt32Array)

Constructs a **PackedInt32Array** as a copy of the given **PackedInt32Array**.

PackedInt32Array **PackedInt32Array**(from: Array)

Constructs a new **PackedInt32Array**. Optionally, you can pass in a generic Array that will be converted.

Method Descriptions

bool **append**(value: int) 🔗

Appends an element at the end of the array (alias of push_back).

---

void **append_array**(array: PackedInt32Array) 🔗

Appends a **PackedInt32Array** at the end of this array.

---

int **bsearch**(value: int, before: bool = true) 🔗

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a `before` specifier can be passed. If `false`, the returned index comes after all existing entries of the value in the array.

Note: Calling bsearch on an unsorted array results in unexpected behavior.

---

void **clear**() 🔗

Clears the array. This is equivalent to using resize with a size of `0`.

---

int **count**(value: int) const 🔗

Returns the number of times an element is in the array.

PackedInt32Array **duplicate**() 🔗

Creates a copy of the array, and returns it.

void **fill**(value: int)

Assigns the given value to all elements in the array. This can typically be used together with resize to create an array with a given size and initialized elements.

int **find**(value: int, from: int = 0) const

Searches the array for a value and returns its index or `-1` if not found. Optionally, the initial search index can be passed.

bool **has**(value: int) const

Returns `true` if the array contains `value`.

int **insert**(at_index: int, value: int)

Inserts a new integer at a given position in the array. The position must be valid, or at the end of the array (`idx == size()`).

bool **is_empty**() const

Returns `true` if the array is empty.

bool **push_back**(value: int)

Appends a value to the array.

void **remove_at**(index: int) 🔗

Removes an element from the array by index.

---

int **resize**(new_size: int) 🔗

Sets the size of the array. If the array is grown, reserves elements at the end of the array. If the array is shrunk, truncates the array to the new size. Calling resize once and assigning the new values is faster than adding new elements one by one.

---

void **reverse**() 🔗

Reverses the order of the elements in the array.

---

int **rfind**(value: int, from: int = -1) const 🔗

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

---

void **set**(index: int, value: int) 🔗

Changes the integer at the given index.

---

int **size**() const 🔗

Returns the number of elements in the array.

PackedInt32Array **slice**(begin: int, end: int = 2147483647) const 🔗

Returns the slice of the **PackedInt32Array**, from `begin` (inclusive) to `end` (exclusive), as a new **PackedInt32Array**.

The absolute value of `begin` and `end` will be clamped to the array size, so the default value for `end` makes it slice to the size of the array by default (i.e. `arr.slice(1)` is a shorthand for `arr.slice(1, arr.size())`).

If either `begin` or `end` are negative, they will be relative to the end of the array (i.e. `arr.slice(0, -2)` is a shorthand for `arr.slice(0, arr.size() - 2)`).

---

void **sort**() 🔗

Sorts the elements of the array in ascending order.

PackedByteArray **to_byte_array**() const 🔗

Returns a copy of the data converted to a PackedByteArray, where each element have been encoded as 4 bytes.

The size of the new array will be `int32_array.size() * 4`.

Operator Descriptions

bool **operator !=**(right: PackedInt32Array) 🔗

Returns `true` if contents of the arrays differ.

PackedInt32Array **operator +**(right: PackedInt32Array) 🔗

Returns a new **PackedInt32Array** with contents of `right` added at the end of this array. For better performance, consider using append_array instead.

---

bool **operator ==**(right: PackedInt32Array) 🔗

Returns `true` if contents of both arrays are the same, i.e. they have all equal ints at the corresponding indices.

---

int **operator []**(index: int) 🔗

Returns the int at index `index`. Negative indices can be used to access the elements starting from the end. Using index out of array's bounds will result in an error.

Note that int type is 64-bit, unlike the values stored in the array.

PackedInt64Array

A packed array of 64-bit integers.

Description

An array specifically designed to hold 64-bit integer values. Packs data tightly, so it saves memory for large array sizes.

Note: This type stores signed 64-bit integers, which means it can take values in the interval `[-2^63, 2^63 - 1]`, i.e. `[-9223372036854775808, 9223372036854775807]`. Exceeding those bounds will wrap around. If you only need to pack 32-bit integers tightly, see PackedInt32Array for a more memory-friendly alternative.

Differences between packed arrays, typed arrays, and untyped arrays: Packed arrays are generally faster to iterate on and modify compared to a typed array of the same type (e.g. PackedInt32Array versus `Array[int]`). Also, packed arrays consume less memory. As a downside, packed arrays are less flexible as they don't offer as many convenience methods such as Array.map. Typed arrays are in turn faster to iterate on and modify than untyped arrays.

Note: Packed arrays are always passed by reference. To get a copy of an array that can be modified independently of the original array, use duplicate. This is *not* the case for built-in properties and methods. The returned packed array of these are a copies, and changing it will *not* affect the original value. To update a built-in property you need to modify the returned array, and then assign it to the property again.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

PackedInt64Array	PackedInt64Array()
PackedInt64Array	PackedInt64Array(from: PackedInt64Array)
PackedInt64Array	PackedInt64Array(from: Array)

Methods

bool	append(value: int)
void	append_array(array: PackedInt64Array)
int	bsearch(value: int, before: bool = true)
void	clear()
int	count(value: int) const

PackedInt64Array	duplicate()
void	fill(value: int)
int	find(value: int, from: int = 0) const
bool	has(value: int) const
int	insert(at_index: int, value: int)
bool	is_empty() const
bool	push_back(value: int)
void	remove_at(index: int)
int	resize(new_size: int)
void	reverse()
int	rfind(value: int, from: int = -1) const
void	set(index: int, value: int)
int	size() const

PackedInt64Array	slice(begin: int, end: int = 2147483647) const
void	sort()
PackedByteArray	to_byte_array() const

Operators

bool	operator !=(right: PackedInt64Array)
PackedInt64Array	operator +(right: PackedInt64Array)
bool	operator ==(right: PackedInt64Array)
int	operator [](index: int)

Constructor Descriptions

PackedInt64Array **PackedInt64Array**() 🔗

Constructs an empty **PackedInt64Array**.

PackedInt64Array **PackedInt64Array**(from: PackedInt64Array)

Constructs a **PackedInt64Array** as a copy of the given **PackedInt64Array**.

PackedInt64Array **PackedInt64Array**(from: Array)

Constructs a new **PackedInt64Array**. Optionally, you can pass in a generic Array that will be converted.

Method Descriptions

bool **append**(value: int)

Appends an element at the end of the array (alias of push_back).

---

void **append_array**(array: PackedInt64Array)

Appends a **PackedInt64Array** at the end of this array.

---

int **bsearch**(value: int, before: bool = true)

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a `before` specifier can be passed. If `false`, the returned index comes after all existing entries of the value in the array.

Note: Calling bsearch on an unsorted array results in unexpected behavior.

---

void **clear**()

Clears the array. This is equivalent to using resize with a size of `0`.

int **count**(value: int) const

Returns the number of times an element is in the array.

PackedInt64Array **duplicate**()

Creates a copy of the array, and returns it.

void **fill**(value: int)

Assigns the given value to all elements in the array. This can typically be used together with resize to create an array with a given size and initialized elements.

int **find**(value: int, from: int = 0) const

Searches the array for a value and returns its index or `-1` if not found. Optionally, the initial search index can be passed.

bool **has**(value: int) const

Returns `true` if the array contains `value`.

int **insert**(at_index: int, value: int)

Inserts a new integer at a given position in the array. The position must be valid, or at the end of the array (`idx == size()`).

bool **is_empty**() const

Returns `true` if the array is empty.

---

bool **push_back**(value: int)

Appends a value to the array.

---

void **remove_at**(index: int)

Removes an element from the array by index.

---

int **resize**(new_size: int)

Sets the size of the array. If the array is grown, reserves elements at the end of the array. If the array is shrunk, truncates the array to the new size. Calling resize once and assigning the new values is faster than adding new elements one by one.

---

void **reverse**()

Reverses the order of the elements in the array.

---

int **rfind**(value: int, from: int = -1) const

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

---

void **set**(index: int, value: int)

Changes the integer at the given index.

---

int **size**() const

Returns the number of elements in the array.

PackedInt64Array **slice**(begin: int, end: int = 2147483647) const

Returns the slice of the **PackedInt64Array**, from `begin` (inclusive) to `end` (exclusive), as a new **PackedInt64Array**.

The absolute value of `begin` and `end` will be clamped to the array size, so the default value for `end` makes it slice to the size of the array by default (i.e. `arr.slice(1)` is a shorthand for `arr.slice(1, arr.size())`).

If either `begin` or `end` are negative, they will be relative to the end of the array (i.e. `arr.slice(0, -2)` is a shorthand for `arr.slice(0, arr.size() - 2)`).

---

void **sort**()

Sorts the elements of the array in ascending order.

PackedByteArray **to_byte_array**() const

Returns a copy of the data converted to a PackedByteArray, where each element have been encoded as 8 bytes.

The size of the new array will be `int64_array.size() * 8`.

Operator Descriptions

bool **operator !=**(right: PackedInt64Array)

Returns `true` if contents of the arrays differ.

PackedInt64Array **operator +**(right: PackedInt64Array)

Returns a new **PackedInt64Array** with contents of `right` added at the end of this array. For better performance, consider using append_array instead.

---

bool **operator ==**(right: PackedInt64Array)

Returns `true` if contents of both arrays are the same, i.e. they have all equal ints at the corresponding indices.

---

int **operator []**(index: int)

Returns the int at index `index`. Negative indices can be used to access the elements starting from the end. Using index out of array's bounds will result in an error.

PackedStringArray

A packed array of Strings.

Description

An array specifically designed to hold Strings. Packs data tightly, so it saves memory for large array sizes.

If you want to join the strings in the array, use String.join.

```
var string_array = PackedStringArray(["hello", "world"])
var string = " ".join(string_array)
print(string) # "hello world"
```

Differences between packed arrays, typed arrays, and untyped arrays: Packed arrays are generally faster to iterate on and modify compared to a typed array of the same type (e.g. **PackedStringArray** versus `Array[String]`). Also, packed arrays consume less memory. As a downside, packed arrays are less flexible as they don't offer as many convenience methods such as Array.map. Typed arrays are in turn faster to iterate on and modify than untyped arrays.

Note: Packed arrays are always passed by reference. To get a copy of an array that can be modified independently of the original array, use duplicate. This is *not* the case for built-in properties and methods. The returned packed array of these are a copies, and changing it will *not* affect the original value. To update a built-in property you need to modify the returned array, and then assign it to the property again.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Operating System Testing Demo [https://godotengine.org/asset-library/asset/2789]

Constructors

PackedStringArray	PackedStringArray()
PackedStringArray	PackedStringArray(from: PackedStringArray)
PackedStringArray	PackedStringArray(from: Array)

Methods

bool	append(value: String)
void	append_array(array: PackedStringArray)
int	bsearch(value: String, before: bool = true)
void	clear()

int	count(value: String) const
PackedStringArray	duplicate()
void	fill(value: String)
int	find(value: String, from: int = 0) const
bool	has(value: String) const
int	insert(at_index: int, value: String)
bool	is_empty() const
bool	push_back(value: String)
void	remove_at(index: int)
int	resize(new_size: int)
void	reverse()
int	rfind(value: String, from: int = -1) const
void	set(index: int, value: String)

int	size() const
PackedStringArray	slice(begin: int, end: int = 2147483647) const
void	sort()
PackedByteArray	to_byte_array() const

Operators

bool	operator !=(right: PackedStringArray)
PackedStringArray	operator +(right: PackedStringArray)
bool	operator ==(right: PackedStringArray)
String	operator [](index: int)

Constructor Descriptions

PackedStringArray **PackedStringArray**() 🔗

Constructs an empty **PackedStringArray**.

PackedStringArray **PackedStringArray**(from: PackedStringArray)

Constructs a **PackedStringArray** as a copy of the given **PackedStringArray**.

PackedStringArray **PackedStringArray**(from: Array)

Constructs a new **PackedStringArray**. Optionally, you can pass in a generic Array that will be converted.

Method Descriptions

bool **append**(value: String)

Appends an element at the end of the array (alias of push_back).

---

void **append_array**(array: PackedStringArray)

Appends a **PackedStringArray** at the end of this array.

---

int **bsearch**(value: String, before: bool = true)

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a `before` specifier can be passed. If `false`, the returned index comes after all existing entries of the value in the array.

Note: Calling bsearch on an unsorted array results in unexpected behavior.

---

void **clear**()

Clears the array. This is equivalent to using resize with a size of `0`.

---

int **count**(value: String) const

Returns the number of times an element is in the array.

PackedStringArray **duplicate**()

Creates a copy of the array, and returns it.

---

void **fill**(value: String)

Assigns the given value to all elements in the array. This can typically be used together with resize to create an array with a given size and initialized elements.

---

int **find**(value: String, from: int = 0) const

Searches the array for a value and returns its index or `-1` if not found. Optionally, the initial search index can be passed.

---

bool **has**(value: String) const

Returns `true` if the array contains `value`.

---

int **insert**(at_index: int, value: String)

Inserts a new element at a given position in the array. The position must be valid, or at the end of the array (`idx == size()`).

---

bool **is_empty**() const

Returns `true` if the array is empty.

---

bool **push_back**(value: String)

Appends a string element at end of the array.

---

void **remove_at**(index: int)

Removes an element from the array by index.

---

int **resize**(new_size: int)

Sets the size of the array. If the array is grown, reserves elements at the end of the array. If the array is shrunk, truncates the array to the new size. Calling resize once and assigning the new values is faster than adding new elements one by one.

---

void **reverse**()

Reverses the order of the elements in the array.

---

int **rfind**(value: String, from: int = -1) const

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

---

void **set**(index: int, value: String)

Changes the String at the given index.

---

int **size**() const 🔗

Returns the number of elements in the array.

PackedStringArray **slice**(begin: int, end: int = 2147483647) const 🔗

Returns the slice of the **PackedStringArray**, from `begin` (inclusive) to `end` (exclusive), as a new **PackedStringArray**.

The absolute value of `begin` and `end` will be clamped to the array size, so the default value for `end` makes it slice to the size of the array by default (i.e. `arr.slice(1)` is a shorthand for `arr.slice(1, arr.size())`).

If either `begin` or `end` are negative, they will be relative to the end of the array (i.e. `arr.slice(0, -2)` is a shorthand for `arr.slice(0, arr.size() - 2)`).

---

void **sort**() 🔗

Sorts the elements of the array in ascending order.

PackedByteArray **to_byte_array**() const 🔗

Returns a PackedByteArray with each string encoded as bytes.

Operator Descriptions

bool **operator !=**(right: PackedStringArray) 🔗

Returns `true` if contents of the arrays differ.

PackedStringArray **operator +**(right: PackedStringArray) 🔗

Returns a new **PackedStringArray** with contents of `right` added at the end of this array. For better performance, consider using append_array instead.

---

bool **operator ==**(right: PackedStringArray) 🔗

Returns `true` if contents of both arrays are the same, i.e. they have all equal Strings at the corresponding indices.

String **operator []**(index: int) 🔗

Returns the String at index `index`. Negative indices can be used to access the elements starting from the end. Using index out of array's bounds will result in an error.

PackedVector2Array

A packed array of Vector2s.

Description

An array specifically designed to hold Vector2. Packs data tightly, so it saves memory for large array sizes.

Differences between packed arrays, typed arrays, and untyped arrays: Packed arrays are generally faster to iterate on and modify compared to a typed array of the same type (e.g. PackedVector3Array versus `Array[Vector2]`). Also, packed arrays consume less memory. As a downside, packed arrays are less flexible as they don't offer as many convenience methods such as Array.map. Typed arrays are in turn faster to iterate on and modify than untyped arrays.

Note: Packed arrays are always passed by reference. To get a copy of an array that can be modified independently of the original array, use duplicate. This is *not* the case for built-in properties and methods. The returned packed array of these are a copies, and changing it will *not* affect the original value. To update a built-in property you need to modify the returned array, and then assign it to the property again.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Grid-based Navigation with AStarGrid2D Demo [https://godotengine.org/asset-library/asset/2723]

Constructors

PackedVector2Array	PackedVector2Array()
PackedVector2Array	PackedVector2Array(from: PackedVector2Array)
PackedVector2Array	PackedVector2Array(from: Array)

Methods

bool	append(value: Vector2)
void	append_array(array: PackedVector2Array)
int	bsearch(value: Vector2, before: bool = true)
void	clear()
int	count(value: Vector2) const

PackedVector2Array	duplicate()
void	fill(value: Vector2)
int	find(value: Vector2, from: int = 0) const
bool	has(value: Vector2) const
int	insert(at_index: int, value: Vector2)
bool	is_empty() const
bool	push_back(value: Vector2)
void	remove_at(index: int)
int	resize(new_size: int)
void	reverse()
int	rfind(value: Vector2, from: int = -1) const
void	set(index: int, value: Vector2)
int	size() const

PackedVector2Array	slice(begin: int, end: int = 2147483647) const
void	sort()
PackedByteArray	to_byte_array() const

Operators

bool	operator !=(right: PackedVector2Array)
PackedVector2Array	operator *(right: Transform2D)
PackedVector2Array	operator +(right: PackedVector2Array)
bool	operator ==(right: PackedVector2Array)
Vector2	operator [](index: int)

Constructor Descriptions

PackedVector2Array **PackedVector2Array**() 🔗

Constructs an empty **PackedVector2Array**.

PackedVector2Array **PackedVector2Array**(from: PackedVector2Array)

Constructs a **PackedVector2Array** as a copy of the given **PackedVector2Array**.

PackedVector2Array **PackedVector2Array**(from: Array)

Constructs a new **PackedVector2Array**. Optionally, you can pass in a generic Array that will be converted.

Note: When initializing a **PackedVector2Array** with elements, it must be initialized with an Array of Vector2 values:

```
var array = PackedVector2Array([Vector2(12, 34), Vector2(56, 78)])
```

Method Descriptions

bool **append**(value: Vector2)

Appends an element at the end of the array (alias of push_back).

void **append_array**(array: PackedVector2Array)

Appends a **PackedVector2Array** at the end of this array.

int **bsearch**(value: Vector2, before: bool = true)

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a `before` specifier can be passed. If `false`, the returned index comes after all existing entries of the value in the array.

Note: Calling bsearch on an unsorted array results in unexpected behavior.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

---

void **clear**() 🔗

Clears the array. This is equivalent to using resize with a size of `0`.

---

int **count**(value: Vector2) const 🔗

Returns the number of times an element is in the array.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

PackedVector2Array **duplicate**() 🔗

Creates a copy of the array, and returns it.

---

void **fill**(value: Vector2) 🔗

Assigns the given value to all elements in the array. This can typically be used together with resize to create an array with a given size and initialized elements.

---

int **find**(value: Vector2, from: int = 0) const 🔗

Searches the array for a value and returns its index or `-1` if not found. Optionally, the initial search index can be passed.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

---

bool **has**(value: Vector2) const 🔗

Returns `true` if the array contains `value`.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

---

int **insert**(at_index: int, value: Vector2) 🔗

Inserts a new element at a given position in the array. The position must be valid, or at the end of the array (`idx == size()`).

---

bool **is_empty**() const 🔗

Returns `true` if the array is empty.

---

bool **push_back**(value: Vector2) 🔗

Inserts a Vector2 at the end.

---

void **remove_at**(index: int) 🔗

Removes an element from the array by index.

---

int **resize**(new_size: int) 🔗

Sets the size of the array. If the array is grown, reserves elements at the end of the array. If the array is shrunk, truncates the array to the new size. Calling resize once and assigning the new values is faster than adding new elements one by one.

---

void **reverse**() 🔗

Reverses the order of the elements in the array.

---

int **rfind**(value: Vector2, from: int = -1) const 🔗

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

---

void **set**(index: int, value: Vector2) 🔗

Changes the Vector2 at the given index.

---

int **size**() const 🔗

Returns the number of elements in the array.

PackedVector2Array **slice**(begin: int, end: int = 2147483647) const 🔗

Returns the slice of the **PackedVector2Array**, from `begin` (inclusive) to `end` (exclusive), as a new **PackedVector2Array**.

The absolute value of `begin` and `end` will be clamped to the array size, so the default value for `end` makes it slice to the size of the array by default (i.e. `arr.slice(1)` is a shorthand for `arr.slice(1, arr.size())`).

If either `begin` or `end` are negative, they will be relative to the end of the array (i.e. `arr.slice(0, -2)` is a shorthand for `arr.slice(0, arr.size() - 2)`).

---

void **sort**() 🔗

Sorts the elements of the array in ascending order.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

PackedByteArray **to_byte_array**() const 🔗

Returns a PackedByteArray with each vector encoded as bytes.

Operator Descriptions

bool **operator !=**(right: PackedVector2Array) 🔗

Returns `true` if contents of the arrays differ.

PackedVector2Array **operator ***(right: Transform2D)

Returns a new **PackedVector2Array** with all vectors in this array inversely transformed (multiplied) by the given Transform2D transformation matrix, under the assumption that the transformation basis is orthonormal (i.e. rotation/reflection is fine, scaling/skew is not).

`array * transform` is equivalent to `transform.inverse() * array`. See Transform2D.inverse.

For transforming by inverse of an affine transformation (e.g. with scaling) `transform.affine_inverse() * array` can be used instead. See Transform2D.affine_inverse.

PackedVector2Array **operator +**(right: PackedVector2Array)

Returns a new **PackedVector2Array** with contents of `right` added at the end of this array. For better performance, consider using append_array instead.

---

bool **operator ==**(right: PackedVector2Array)

Returns `true` if contents of both arrays are the same, i.e. they have all equal Vector2s at the corresponding indices.

Vector2 **operator []**(index: int)

Returns the Vector2 at index `index`. Negative indices can be used to access the elements starting from the end. Using index out of array's bounds will result in an error.

PackedVector3Array

A packed array of Vector3s.

Description

An array specifically designed to hold Vector3. Packs data tightly, so it saves memory for large array sizes.

Differences between packed arrays, typed arrays, and untyped arrays: Packed arrays are generally faster to iterate on and modify compared to a typed array of the same type (e.g. **PackedVector3Array** versus `Array[Vector3]`). Also, packed arrays consume less memory. As a downside, packed arrays are less flexible as they don't offer as many convenience methods such as Array.map. Typed arrays are in turn faster to iterate on and modify than untyped arrays.

Note: Packed arrays are always passed by reference. To get a copy of an array that can be modified independently of the original array, use duplicate. This is *not* the case for built-in properties and methods. The returned packed array of these are a copies, and changing it will *not* affect the original value. To update a built-in property you need to modify the returned array, and then assign it to the property again.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

PackedVector3Array	PackedVector3Array()
PackedVector3Array	PackedVector3Array(from: PackedVector3Array)
PackedVector3Array	PackedVector3Array(from: Array)

Methods

bool	append(value: Vector3)
void	append_array(array: PackedVector3Array)
int	bsearch(value: Vector3, before: bool = true)
void	clear()
int	count(value: Vector3) const
PackedVector3Array	duplicate()
void	fill(value: Vector3)

int	find(value: Vector3, from: int = 0) const
bool	has(value: Vector3) const
int	insert(at_index: int, value: Vector3)
bool	is_empty() const
bool	push_back(value: Vector3)
void	remove_at(index: int)
int	resize(new_size: int)
void	reverse()
int	rfind(value: Vector3, from: int = -1) const
void	set(index: int, value: Vector3)
int	size() const
PackedVector3Array	slice(begin: int, end: int = 2147483647) const
void	sort()

PackedByteArray	to_byte_array() const

Operators

bool	operator !=(right: PackedVector3Array)
PackedVector3Array	operator *(right: Transform3D)
PackedVector3Array	operator +(right: PackedVector3Array)
bool	operator ==(right: PackedVector3Array)
Vector3	operator [](index: int)

Constructor Descriptions

PackedVector3Array **PackedVector3Array**() 🔗

Constructs an empty **PackedVector3Array**.

PackedVector3Array **PackedVector3Array**(from: PackedVector3Array)

Constructs a **PackedVector3Array** as a copy of the given **PackedVector3Array**.

PackedVector3Array **PackedVector3Array**(from: Array)

Constructs a new **PackedVector3Array**. Optionally, you can pass in a generic Array that will be converted.

Note: When initializing a **PackedVector3Array** with elements, it must be initialized with an Array of Vector3 values:

```
var array = PackedVector3Array([Vector3(12, 34, 56),
Vector3(78, 90, 12)])
```

Method Descriptions

bool **append**(value: Vector3)

Appends an element at the end of the array (alias of push_back).

---

void **append_array**(array: PackedVector3Array)

Appends a **PackedVector3Array** at the end of this array.

---

int **bsearch**(value: Vector3, before: bool = true)

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a `before` specifier can be passed. If `false`, the returned index comes after all existing entries of the value in the array.

Note: Calling bsearch on an unsorted array results in unexpected behavior.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

---

void **clear**()

Clears the array. This is equivalent to using resize with a size of `0`.

---

int **count**(value: Vector3) const

Returns the number of times an element is in the array.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

PackedVector3Array **duplicate**()

Creates a copy of the array, and returns it.

---

void **fill**(value: Vector3)

Assigns the given value to all elements in the array. This can typically be used together with resize to create an array with a given size and initialized elements.

---

int **find**(value: Vector3, from: int = 0) const

Searches the array for a value and returns its index or `-1` if not found. Optionally, the initial search index can be passed.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

---

bool **has**(value: Vector3) const

Returns `true` if the array contains `value`.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

---

int **insert**(at_index: int, value: Vector3)

Inserts a new element at a given position in the array. The position must be valid, or at the end of the array (`idx == size()`).

---

bool **is_empty**() const

Returns `true` if the array is empty.

---

bool **push_back**(value: Vector3)

Inserts a Vector3 at the end.

---

void **remove_at**(index: int)

Removes an element from the array by index.

---

int **resize**(new_size: int)

Sets the size of the array. If the array is grown, reserves elements at the end of the array. If the array is shrunk, truncates the array to the new size. Calling resize once and assigning the new values is faster than adding new elements one by one.

void **reverse**()

Reverses the order of the elements in the array.

int **rfind**(value: Vector3, from: int = -1) const

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

void **set**(index: int, value: Vector3)

Changes the Vector3 at the given index.

int **size**() const

Returns the number of elements in the array.

PackedVector3Array **slice**(begin: int, end: int = 2147483647) const

Returns the slice of the **PackedVector3Array**, from `begin` (inclusive) to `end` (exclusive), as a new **PackedVector3Array**.

The absolute value of `begin` and `end` will be clamped to the array size, so the default value for `end` makes it slice to the size of the array by default (i.e. `arr.slice(1)` is a shorthand for `arr.slice(1, arr.size())`).

If either `begin` or `end` are negative, they will be relative to the end of the array (i.e. `arr.slice(0, -2)` is a shorthand for `arr.slice(0, arr.size() - 2)`).

---

void **sort**()

Sorts the elements of the array in ascending order.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

PackedByteArray **to_byte_array**() const

Returns a PackedByteArray with each vector encoded as bytes.

Operator Descriptions

bool **operator !=**(right: PackedVector3Array)

Returns `true` if contents of the arrays differ.

PackedVector3Array **operator ***(right: Transform3D)

Returns a new **PackedVector3Array** with all vectors in this array inversely transformed (multiplied) by the given Transform3D transformation matrix, under the assumption that the transformation basis is orthonormal (i.e. rotation/reflection is fine, scaling/skew is not).

`array * transform` is equivalent to `transform.inverse() * array`. See Transform3D.inverse.

For transforming by inverse of an affine transformation (e.g. with scaling) `transform.affine_inverse() * array` can be used instead. See Transform3D.affine_inverse.

PackedVector3Array **operator +**(right: PackedVector3Array) 🔗

Returns a new **PackedVector3Array** with contents of `right` added at the end of this array. For better performance, consider using append_array instead.

---

bool **operator ==**(right: PackedVector3Array) 🔗

Returns `true` if contents of both arrays are the same, i.e. they have all equal Vector3s at the corresponding indices.

Vector3 **operator []**(index: int) 🔗

Returns the Vector3 at index `index`. Negative indices can be used to access the elements starting from the end. Using index out of array's bounds will result in an error.

PackedVector4Array

A packed array of Vector4s.

Description

An array specifically designed to hold Vector4. Packs data tightly, so it saves memory for large array sizes.

Note: Packed arrays are always passed by reference. To get a copy of an array that can be modified independently of the original array, use duplicate. This is *not* the case for built-in properties and methods. The returned packed array of these are a copies, and changing it will *not* affect the original value. To update a built-in property you need to modify the returned array, and then assign it to the property again.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

PackedVector4Array	PackedVector4Array()
PackedVector4Array	PackedVector4Array(from: PackedVector4Array)

PackedVector4Array	PackedVector4Array(from: Array)

Methods

bool	append(value: Vector4)
void	append_array(array: PackedVector4Array)
int	bsearch(value: Vector4, before: bool = true)
void	clear()
int	count(value: Vector4) const
PackedVector4Array	duplicate()
void	fill(value: Vector4)
int	find(value: Vector4, from: int = 0) const
bool	has(value: Vector4) const
int	insert(at_index: int, value: Vector4)

bool	is_empty() const
bool	push_back(value: Vector4)
void	remove_at(index: int)
int	resize(new_size: int)
void	reverse()
int	rfind(value: Vector4, from: int = -1) const
void	set(index: int, value: Vector4)
int	size() const
PackedVector4Array	slice(begin: int, end: int = 2147483647) const
void	sort()
PackedByteArray	to_byte_array() const

Operators

bool	operator !=(right: PackedVector4Array)
PackedVector4Array	operator +(right: PackedVector4Array)
bool	operator ==(right: PackedVector4Array)
Vector4	operator [](index: int)

Constructor Descriptions

PackedVector4Array **PackedVector4Array**() 🔗

Constructs an empty **PackedVector4Array**.

PackedVector4Array **PackedVector4Array**(from: PackedVector4Array)

Constructs a **PackedVector4Array** as a copy of the given **PackedVector4Array**.

PackedVector4Array **PackedVector4Array**(from: Array)

Constructs a new **PackedVector4Array**. Optionally, you can pass in a generic Array that will be converted.

Note: When initializing a **PackedVector4Array** with elements, it must be initialized with an Array of Vector4 values:

```
var array = PackedVector4Array([Vector4(12, 34, 56, 78),
Vector4(90, 12, 34, 56)])
```

Method Descriptions

bool **append**(value: Vector4) 🔗

Appends an element at the end of the array (alias of push_back).

void **append_array**(array: PackedVector4Array) 🔗

Appends a **PackedVector4Array** at the end of this array.

int **bsearch**(value: Vector4, before: bool = true) 🔗

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a `before` specifier can be passed. If `false`, the returned index comes after all existing entries of the value in the array.

Note: Calling bsearch on an unsorted array results in unexpected behavior.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

void **clear**() 🔗

Clears the array. This is equivalent to using resize with a size of `0`.

int **count**(value: Vector4) const

Returns the number of times an element is in the array.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

PackedVector4Array **duplicate**()

Creates a copy of the array, and returns it.

---

void **fill**(value: Vector4)

Assigns the given value to all elements in the array. This can typically be used together with resize to create an array with a given size and initialized elements.

---

int **find**(value: Vector4, from: int = 0) const

Searches the array for a value and returns its index or `-1` if not found. Optionally, the initial search index can be passed.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

---

bool **has**(value: Vector4) const

Returns `true` if the array contains `value`.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

---

int **insert**(at_index: int, value: Vector4)

Inserts a new element at a given position in the array. The position must be valid, or at the end of the array (`idx == size()`).

---

bool **is_empty**() const

Returns `true` if the array is empty.

---

bool **push_back**(value: Vector4)

Inserts a Vector4 at the end.

---

void **remove_at**(index: int)

Removes an element from the array by index.

---

int **resize**(new_size: int)

Sets the size of the array. If the array is grown, reserves elements at the end of the array. If the array is shrunk, truncates the array to the new size.

---

void **reverse**()

Reverses the order of the elements in the array.

---

int **rfind**(value: Vector4, from: int = -1) const

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

---

void **set**(index: int, value: Vector4)

Changes the Vector4 at the given index.

---

int **size**() const

Returns the number of elements in the array.

PackedVector4Array **slice**(begin: int, end: int = 2147483647) const

Returns the slice of the **PackedVector4Array**, from `begin` (inclusive) to `end` (exclusive), as a new **PackedVector4Array**.

The absolute value of `begin` and `end` will be clamped to the array size, so the default value for `end` makes it slice to the size of the array by default (i.e. `arr.slice(1)` is a shorthand for `arr.slice(1, arr.size())`).

If either `begin` or `end` are negative, they will be relative to the end of the array (i.e. `arr.slice(0, -2)` is a shorthand for `arr.slice(0, arr.size() - 2)`).

---

void **sort**()

Sorts the elements of the array in ascending order.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this method may not be accurate if NaNs are included.

PackedByteArray **to_byte_array**() const 🔗

Returns a PackedByteArray with each vector encoded as bytes.

Operator Descriptions

bool **operator !=**(right: PackedVector4Array) 🔗

Returns `true` if contents of the arrays differ.

PackedVector4Array **operator +**(right: PackedVector4Array) 🔗

Returns a new **PackedVector4Array** with contents of `right` added at the end of this array. For better performance, consider using append_array instead.

---

bool **operator ==**(right: PackedVector4Array) 🔗

Returns `true` if contents of both arrays are the same, i.e. they have all equal Vector4s at the corresponding indices.

Vector4 **operator []**(index: int) 🔗

Returns the Vector4 at index `index`. Negative indices can be used to access the elements starting from the end. Using index out of array's bounds will result in an error.

Plane

A plane in Hessian normal form.

Description

Represents a normalized plane equation. normal is the normal of the plane (a, b, c normalized), and d is the distance from the origin to the plane (in the direction of "normal"). "Over" or "Above" the plane is considered the side of the plane towards where the normal is pointing.

Tutorials

- Math documentation index

Properties

float	d	`0.0`
Vector3	normal	`Vector3(0, 0, 0)`
float	x	`0.0`
float	y	`0.0`

float	z	`0.0`

Constructors

Plane	Plane()
Plane	Plane(from: Plane)
Plane	Plane(a: float, b: float, c: float, d: float)
Plane	Plane(normal: Vector3)
Plane	Plane(normal: Vector3, d: float)
Plane	Plane(normal: Vector3, point: Vector3)
Plane	Plane(point1: Vector3, point2: Vector3, point3: Vector3)

Methods

float	distance_to(point: Vector3) const
Vector3	get_center() const

bool	has_point(point: Vector3, tolerance: float = 1e-05) const
Variant	intersect_3(b: Plane, c: Plane) const
Variant	intersects_ray(from: Vector3, dir: Vector3) const
Variant	intersects_segment(from: Vector3, to: Vector3) const
bool	is_equal_approx(to_plane: Plane) const
bool	is_finite() const
bool	is_point_over(point: Vector3) const
Plane	normalized() const
Vector3	project(point: Vector3) const

Operators

bool	operator !=(right: Plane)
Plane	operator *(right: Transform3D)

bool	operator ==(right: Plane)
Plane	operator unary+()
Plane	operator unary-()

Constants

PLANE_YZ = `Plane(1, 0, 0, 0)`

A plane that extends in the Y and Z axes (normal vector points +X).

PLANE_XZ = `Plane(0, 1, 0, 0)`

A plane that extends in the X and Z axes (normal vector points +Y).

PLANE_XY = `Plane(0, 0, 1, 0)`

A plane that extends in the X and Y axes (normal vector points +Z).

Property Descriptions

float **d** = `0.0`

The distance from the origin to the plane, expressed in terms of normal (according to its direction and magnitude). Actual absolute distance from the origin to the plane can be calculated as `abs(d) / normal.length()` (if normal has zero length then this **Plane** does not represent a valid plane).

In the scalar equation of the plane `ax + by + cz = d`, this is `d`, while the `(a, b, c)` coordinates are represented by the normal property.

Vector3 **normal** = `Vector3(0, 0, 0)` 🔗

The normal of the plane, typically a unit vector. Shouldn't be a zero vector as **Plane** with such normal does not represent a valid plane.

In the scalar equation of the plane `ax + by + cz = d`, this is the vector `(a, b, c)`, where `d` is the d property.

---

float **x** = `0.0` 🔗

The X component of the plane's normal vector.

---

float **y** = `0.0` 🔗

The Y component of the plane's normal vector.

---

float **z** = `0.0` 🔗

The Z component of the plane's normal vector.

Constructor Descriptions

Plane **Plane**() 🔗

Constructs a default-initialized **Plane** with all components set to `0`.

Plane **Plane**(from: Plane)

Constructs a **Plane** as a copy of the given **Plane**.

Plane **Plane**(a: float, b: float, c: float, d: float)

Creates a plane from the four parameters. The three components of the resulting plane's normal are `a`, `b` and `c`, and the plane has a distance of `d` from the origin.

Plane **Plane**(normal: Vector3)

Creates a plane from the normal vector. The plane will intersect the origin.

The `normal` of the plane must be a unit vector.

Plane **Plane**(normal: Vector3, d: float)

Creates a plane from the normal vector and the plane's distance from the origin.

The `normal` of the plane must be a unit vector.

Plane **Plane**(normal: Vector3, point: Vector3)

Creates a plane from the normal vector and a point on the plane.

The `normal` of the plane must be a unit vector.

Plane **Plane**(point1: Vector3, point2: Vector3, point3: Vector3)

Creates a plane from the three points, given in clockwise order.

Method Descriptions

float **distance_to**(point: Vector3) const 🔗

Returns the shortest distance from the plane to the position `point`. If the point is above the plane, the distance will be positive. If below, the distance will be negative.

Vector3 **get_center**() const 🔗

Returns the center of the plane.

---

bool **has_point**(point: Vector3, tolerance: float = 1e-05) const 🔗

Returns `true` if `point` is inside the plane. Comparison uses a custom minimum `tolerance` threshold.

Variant **intersect_3**(b: Plane, c: Plane) const 🔗

Returns the intersection point of the three planes `b`, `c` and this plane. If no intersection is found, `null` is returned.

Variant **intersects_ray**(from: Vector3, dir: Vector3) const 🔗

Returns the intersection point of a ray consisting of the position `from` and the direction normal `dir` with this plane. If no intersection is found, `null` is returned.

Variant **intersects_segment**(from: Vector3, to: Vector3) const 🔗

Returns the intersection point of a segment from position `from` to position `to` with this plane. If no intersection is found, `null` is returned.

bool **is_equal_approx**(to_plane: Plane) const 🔗

Returns `true` if this plane and `to_plane` are approximately equal, by running @GlobalScope.is_equal_approx on each component.

bool **is_finite**() const 🔗

Returns `true` if this plane is finite, by calling @GlobalScope.is_finite on each component.

bool **is_point_over**(point: Vector3) const 🔗

Returns `true` if `point` is located above the plane.

Plane **normalized**() const 🔗

Returns a copy of the plane, with normalized normal (so it's a unit vector). Returns `Plane(0, 0, 0, 0)` if normal can't be normalized (it has zero length).

Vector3 **project**(point: Vector3) const 🔗

Returns the orthogonal projection of `point` into a point in the plane.

Operator Descriptions

bool **operator !=**(right: Plane)

Returns `true` if the planes are not equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Plane **operator ***(right: Transform3D)

Inversely transforms (multiplies) the **Plane** by the given Transform3D transformation matrix.

`plane * transform` is equivalent to `transform.affine_inverse() * plane`. See Transform3D.affine_inverse.

---

bool **operator ==**(right: Plane)

Returns `true` if the planes are exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Plane **operator unary+**()

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

Plane **operator unary-**()

Returns the negative value of the **Plane**. This is the same as writing `Plane(-p.normal, -p.d)`. This operation flips the direction of the normal vector and also flips the distance value, resulting in a Plane that is in the same place, but facing the opposite direction.

Projection

A 4×4 matrix for 3D projective transformations.

Description

A 4×4 matrix used for 3D projective transformations. It can represent transformations such as translation, rotation, scaling, shearing, and perspective division. It consists of four Vector4 columns.

For purely linear transformations (translation, rotation, and scale), it is recommended to use Transform3D, as it is more performant and requires less memory.

Used internally as Camera3D's projection matrix.

> **Note**
>
> There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Properties

Vector4	w	`Vector4(0, 0, 0, 1)`
Vector4	x	`Vector4(1, 0, 0, 0)`

Vector4	y	`Vector4(0, 1, 0, 0)`
Vector4	z	`Vector4(0, 0, 1, 0)`

Constructors

Projection	Projection()
Projection	Projection(from: Projection)
Projection	Projection(from: Transform3D)
Projection	Projection(x_axis: Vector4, y_axis: Vector4, z_axis: Vector4, w_axis: Vector4)

Methods

Projection	create_depth_correction(flip_y: bool) static
Projection	create_fit_aabb(aabb: AABB) static
Projection	create_for_hmd(eye: int, aspect: float, intraocular_dist: float, display_width: float, display_to_lens: float, oversample: float, z_near: float, z_far: float) static

Projection	create_frustum(left: float, right: float, bottom: float, top: float, z_near: float, z_far: float) static
Projection	create_frustum_aspect(size: float, aspect: float, offset: Vector2, z_near: float, z_far: float, flip_fov: bool = false) static
Projection	create_light_atlas_rect(rect: Rect2) static
Projection	create_orthogonal(left: float, right: float, bottom: float, top: float, z_near: float, z_far: float) static
Projection	create_orthogonal_aspect(size: float, aspect: float, z_near: float, z_far: float, flip_fov: bool = false) static
Projection	create_perspective(fovy: float, aspect: float, z_near: float, z_far: float, flip_fov: bool = false) static
Projection	create_perspective_hmd(fovy: float, aspect: float, z_near: float, z_far: float, flip_fov: bool, eye: int, intraocular_dist: float, convergence_dist: float) static
float	determinant() const
Projection	flipped_y() const
float	get_aspect() const

Vector2	get_far_plane_half_extents() const
float	get_fov() const
float	get_fovy(fovx: float, aspect: float) static
float	get_lod_multiplier() const
int	get_pixels_per_meter(for_pixel_width: int) const
Plane	get_projection_plane(plane: int) const
Vector2	get_viewport_half_extents() const
float	get_z_far() const
float	get_z_near() const
Projection	inverse() const
bool	is_orthogonal() const
Projection	jitter_offseted(offset: Vector2) const
Projection	perspective_znear_adjusted(new_znear: float) const

Operators

bool	operator !=(right: Projection)
Projection	operator *(right: Projection)
Vector4	operator *(right: Vector4)
bool	operator ==(right: Projection)
Vector4	operator [](index: int)

Constants

PLANE_NEAR = `0` 🔗

The index value of the projection's near clipping plane.

PLANE_FAR = `1` 🔗

The index value of the projection's far clipping plane.

PLANE_LEFT = `2` 🔗

The index value of the projection's left clipping plane.

PLANE_TOP = `3` 🔗

The index value of the projection's top clipping plane.

PLANE_RIGHT = `4` 🔗

The index value of the projection's right clipping plane.

PLANE_BOTTOM = `5` 🔗

The index value of the projection bottom clipping plane.

IDENTITY = `Projection(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1)` 🔗

A **Projection** with no transformation defined. When applied to other data structures, no transformation is performed.

ZERO = `Projection(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)` 🔗

A **Projection** with all values initialized to 0. When applied to other data structures, they will be zeroed.

Property Descriptions

Vector4 **w** = `Vector4(0, 0, 0, 1)` 🔗

The projection matrix's W vector (column 3). Equivalent to array index `3`.

Vector4 **x** = `Vector4(1, 0, 0, 0)` 🔗

The projection matrix's X vector (column 0). Equivalent to array index `0`.

Vector4 **y** = `Vector4(0, 1, 0, 0)`

The projection matrix's Y vector (column 1). Equivalent to array index `1`.

Vector4 **z** = `Vector4(0, 0, 1, 0)`

The projection matrix's Z vector (column 2). Equivalent to array index `2`.

Constructor Descriptions

Projection **Projection**()

Constructs a default-initialized **Projection** set to IDENTITY.

Projection **Projection**(from: Projection)

Constructs a **Projection** as a copy of the given **Projection**.

Projection **Projection**(from: Transform3D)

Constructs a Projection as a copy of the given Transform3D.

Projection **Projection**(x_axis: Vector4, y_axis: Vector4, z_axis: Vector4, w_axis: Vector4)

Constructs a Projection from four Vector4 values (matrix columns).

Method Descriptions

Projection **create_depth_correction**(flip_y: bool) static 🔗

Creates a new **Projection** that projects positions from a depth range of `-1` to `1` to one that ranges from `0` to `1`, and flips the projected positions vertically, according to `flip_y`.

Projection **create_fit_aabb**(aabb: AABB) static 🔗

Creates a new **Projection** that scales a given projection to fit around a given AABB in projection space.

Projection **create_for_hmd**(eye: int, aspect: float, intraocular_dist: float, display_width: float, display_to_lens: float, oversample: float, z_near: float, z_far: float) static 🔗

Creates a new **Projection** for projecting positions onto a head-mounted display with the given X:Y aspect ratio, distance between eyes, display width, distance to lens, oversampling factor, and depth clipping planes.

`eye` creates the projection for the left eye when set to 1, or the right eye when set to 2.

Projection **create_frustum**(left: float, right: float, bottom: float, top: float, z_near: float, z_far: float) static 🔗

Creates a new **Projection** that projects positions in a frustum with the given clipping planes.

Projection **create_frustum_aspect**(size: float, aspect: float, offset: Vector2, z_near: float, z_far: float, flip_fov: bool = false) static

Creates a new **Projection** that projects positions in a frustum with the given size, X:Y aspect ratio, offset, and clipping planes.

`flip_fov` determines whether the projection's field of view is flipped over its diagonal.

Projection **create_light_atlas_rect**(rect: Rect2) static

Creates a new **Projection** that projects positions into the given Rect2.

Projection **create_orthogonal**(left: float, right: float, bottom: float, top: float, z_near: float, z_far: float) static

Creates a new **Projection** that projects positions using an orthogonal projection with the given clipping planes.

Projection **create_orthogonal_aspect**(size: float, aspect: float, z_near: float, z_far: float, flip_fov: bool = false) static

Creates a new **Projection** that projects positions using an orthogonal projection with the given size, X:Y aspect ratio, and clipping planes.

`flip_fov` determines whether the projection's field of view is flipped over its diagonal.

Projection **create_perspective**(fovy: float, aspect: float, z_near: float, z_far: float, flip_fov: bool = false) static

Creates a new **Projection** that projects positions using a perspective projection with the given Y-axis field of view (in degrees), X:Y aspect ratio, and clipping planes.

`flip_fov` determines whether the projection's field of view is flipped over its diagonal.

Projection **create_perspective_hmd**(fovy: float, aspect: float, z_near: float, z_far: float, flip_fov: bool, eye: int, intraocular_dist: float, convergence_dist: float) static

Creates a new **Projection** that projects positions using a perspective projection with the given Y-axis field of view (in degrees), X:Y aspect ratio, and clipping distances. The projection is adjusted for a head-mounted display with the given distance between eyes and distance to a point that can be focused on.

`eye` creates the projection for the left eye when set to 1, or the right eye when set to 2.

`flip_fov` determines whether the projection's field of view is flipped over its diagonal.

---

float **determinant**() const

Returns a scalar value that is the signed factor by which areas are scaled by this matrix. If the sign is negative, the matrix flips the orientation of the area.

The determinant can be used to calculate the invertibility of a matrix or solve linear systems of equations involving the matrix, among other applications.

Projection **flipped_y**() const 🔗

Returns a copy of this **Projection** with the signs of the values of the Y column flipped.

---

float **get_aspect**() const 🔗

Returns the X:Y aspect ratio of this **Projection**'s viewport.

Vector2 **get_far_plane_half_extents**() const 🔗

Returns the dimensions of the far clipping plane of the projection, divided by two.

---

float **get_fov**() const 🔗

Returns the horizontal field of view of the projection (in degrees).

---

float **get_fovy**(fovx: float, aspect: float) static 🔗

Returns the vertical field of view of the projection (in degrees) associated with the given horizontal field of view (in degrees) and aspect ratio.

---

float **get_lod_multiplier**() const 🔗

Returns the factor by which the visible level of detail is scaled by this **Projection**.

---

int **get_pixels_per_meter**(for_pixel_width: int) const 🔗

Returns the number of pixels with the given pixel width displayed per meter, after this **Projection** is applied.

Plane **get_projection_plane**(plane: int) const

Returns the clipping plane of this **Projection** whose index is given by `plane`.

`plane` should be equal to one of PLANE_NEAR, PLANE_FAR, PLANE_LEFT, PLANE_TOP, PLANE_RIGHT, or PLANE_BOTTOM.

Vector2 **get_viewport_half_extents**() const

Returns the dimensions of the viewport plane that this **Projection** projects positions onto, divided by two.

---

float **get_z_far**() const

Returns the distance for this **Projection** beyond which positions are clipped.

---

float **get_z_near**() const

Returns the distance for this **Projection** before which positions are clipped.

Projection **inverse**() const

Returns a **Projection** that performs the inverse of this **Projection**'s projective transformation.

---

bool **is_orthogonal**() const 🔗

Returns `true` if this **Projection** performs an orthogonal projection.

Projection **jitter_offseted**(offset: Vector2) const 🔗

Returns a **Projection** with the X and Y values from the given Vector2 added to the first and second values of the final column respectively.

Projection **perspective_znear_adjusted**(new_znear: float) const 🔗

Returns a **Projection** with the near clipping distance adjusted to be `new_znear`.

Note: The original **Projection** must be a perspective projection.

Operator Descriptions

bool **operator !=**(right: Projection) 🔗

Returns `true` if the projections are not equal.

Note: Due to floating-point precision errors, this may return `true`, even if the projections are virtually equal. An `is_equal_approx` method may be added in a future version of Godot.

Projection **operator ***(right: Projection) 🔗

Returns a **Projection** that applies the combined transformations of this **Projection** and `right`.

Vector4 **operator ***(right: Vector4)

Projects (multiplies) the given Vector4 by this **Projection** matrix.

---

bool **operator ==**(right: Projection)

Returns `true` if the projections are equal.

Note: Due to floating-point precision errors, this may return `false`, even if the projections are virtually equal. An `is_equal_approx` method may be added in a future version of Godot.

Vector4 **operator []**(index: int)

Returns the column of the **Projection** with the given index.

Indices are in the following order: x, y, z, w.

Quaternion

A unit quaternion used for representing 3D rotations.

Description

The **Quaternion** built-in Variant type is a 4D data structure that represents rotation in the form of a Hamilton convention quaternion [https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation]. Compared to the Basis type which can store both rotation and scale, quaternions can *only* store rotation.

A **Quaternion** is composed by 4 floating-point components: w, x, y, and z. These components are very compact in memory, and because of this some operations are more efficient and less likely to cause floating-point errors. Methods such as get_angle, get_axis, and slerp are faster than their Basis counterparts.

For a great introduction to quaternions, see this video by 3Blue1Brown [https://www.youtube.com/watch?v=d4EgbgTm0Bg]. You do not need to know the math behind quaternions, as Godot provides several helper methods that handle it for you. These include slerp and spherical_cubic_interpolate, as well as the `*` operator.

Note: Quaternions must be normalized before being used for rotation (see normalized).

Note: Similarly to Vector2 and Vector3, the components of a quaternion use 32-bit precision by default, unlike float which is always 64-bit. If double precision is needed, compile the engine with the option `precision=double`.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- 3Blue1Brown's video on Quaternions [https://www.youtube.com/watch?v=d4EgbgTm0Bg]
- Online Quaternion Visualization [https://quaternions.online/]
- Using 3D transforms
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]
- Advanced Quaternion Visualization [https://iwatake2222.github.io/rotation_master/rotation_master.html]

Properties

float	w	`1.0`
float	x	`0.0`
float	y	`0.0`
float	z	`0.0`

Constructors

Quaternion	Quaternion()
Quaternion	Quaternion(from: Quaternion)
Quaternion	Quaternion(arc_from: Vector3, arc_to: Vector3)
Quaternion	Quaternion(axis: Vector3, angle: float)
Quaternion	Quaternion(from: Basis)
Quaternion	Quaternion(x: float, y: float, z: float, w: float)

Methods

float	angle_to(to: Quaternion) const
float	dot(with: Quaternion) const
Quaternion	exp() const
Quaternion	from_euler(euler: Vector3) static
float	get_angle() const

Vector3	get_axis() const
Vector3	get_euler(order: int = 2) const
Quaternion	inverse() const
bool	is_equal_approx(to: Quaternion) const
bool	is_finite() const
bool	is_normalized() const
float	length() const
float	length_squared() const
Quaternion	log() const
Quaternion	normalized() const
Quaternion	slerp(to: Quaternion, weight: float) const
Quaternion	slerpni(to: Quaternion, weight: float) const

Quaternion	spherical_cubic_interpolate(b: Quaternion, pre_a: Quaternion, post_b: Quaternion, weight: float) const
Quaternion	spherical_cubic_interpolate_in_time(b: Quaternion, pre_a: Quaternion, post_b: Quaternion, weight: float, b_t: float, pre_a_t: float, post_b_t: float) const

Operators

bool	operator !=(right: Quaternion)
Quaternion	operator *(right: Quaternion)
Vector3	operator *(right: Vector3)
Quaternion	operator *(right: float)
Quaternion	operator *(right: int)
Quaternion	operator +(right: Quaternion)
Quaternion	operator -(right: Quaternion)
Quaternion	operator /(right: float)

Quaternion	operator /(right: int)
bool	operator ==(right: Quaternion)
float	operator [](index: int)
Quaternion	operator unary+()
Quaternion	operator unary-()

Constants

IDENTITY = `Quaternion(0, 0, 0, 1)`

The identity quaternion, representing no rotation. This has the same rotation as Basis.IDENTITY.

If a Vector3 is rotated (multiplied) by this quaternion, it does not change.

Property Descriptions

float **w** = `1.0`

W component of the quaternion. This is the "real" part.

Note: Quaternion components should usually not be manipulated directly.

---

float **x** = `0.0`

X component of the quaternion. This is the value along the "imaginary" `i` axis.

Note: Quaternion components should usually not be manipulated directly.

---

float **y** = `0.0`

Y component of the quaternion. This is the value along the "imaginary" `j` axis.

Note: Quaternion components should usually not be manipulated directly.

---

float **z** = `0.0`

Z component of the quaternion. This is the value along the "imaginary" `k` axis.

Note: Quaternion components should usually not be manipulated directly.

Constructor Descriptions

Quaternion **Quaternion**()

Constructs a **Quaternion** identical to the IDENTITY.

Quaternion **Quaternion**(from: Quaternion)

Constructs a **Quaternion** as a copy of the given **Quaternion**.

Quaternion **Quaternion**(arc_from: Vector3, arc_to: Vector3)

Constructs a **Quaternion** representing the shortest arc between `arc_from` and `arc_to`. These can be imagined as two points intersecting a sphere's surface, with a radius of `1.0`.

Quaternion **Quaternion**(axis: Vector3, angle: float)

Constructs a **Quaternion** representing rotation around the `axis` by the given `angle`, in radians. The axis must be a normalized vector.

Quaternion **Quaternion**(from: Basis)

Constructs a **Quaternion** from the given rotation Basis.

This constructor is faster than Basis.get_rotation_quaternion, but the given basis must be *orthonormalized* (see Basis.orthonormalized). Otherwise, the constructor fails and returns IDENTITY.

Quaternion **Quaternion**(x: float, y: float, z: float, w: float)

Constructs a **Quaternion** defined by the given values.

Note: Only normalized quaternions represent rotation; if these values are not normalized, the new **Quaternion** will not be a valid rotation.

Method Descriptions

float **angle_to**(to: Quaternion) const 🔗

Returns the angle between this quaternion and `to`. This is the magnitude of the angle you would need to rotate by to get from one to the other.

Note: The magnitude of the floating-point error for this method is abnormally high, so methods such as `is_zero_approx` will not work reliably.

---

float **dot**(with: Quaternion) const 🔗

Returns the dot product between this quaternion and `with`.

This is equivalent to `(quat.x * with.x) + (quat.y * with.y) + (quat.z * with.z) + (quat.w * with.w)`.

Quaternion **exp**() const 🔗

Returns the exponential of this quaternion. The rotation axis of the result is the normalized rotation axis of this quaternion, the angle of the result is the length of the vector part of this quaternion.

Quaternion **from_euler**(euler: Vector3) static 🔗

Constructs a new **Quaternion** from the given Vector3 of Euler angles [https://en.wikipedia.org/wiki/Euler_angles], in radians. This method always uses the YXZ convention (@GlobalScope.EULER_ORDER_YXZ).

---

float **get_angle**() const 🔗

Returns the angle of the rotation represented by this quaternion.

Note: The quaternion must be normalized.

Vector3 **get_axis**() const 🔗

Returns the rotation axis of the rotation represented by this quaternion.

Vector3 **get_euler**(order: int = 2) const 🔗

Returns this quaternion's rotation as a Vector3 of Euler angles [https://en.wikipedia.org/wiki/Euler_angles], in radians.

The order of each consecutive rotation can be changed with `order` (see EulerOrder constants). By default, the YXZ convention is used (@GlobalScope.EULER_ORDER_YXZ): Z (roll) is calculated first, then X (pitch), and lastly Y (yaw). When using the opposite method from_euler, this order is reversed.

Quaternion **inverse**() const 🔗

Returns the inverse version of this quaternion, inverting the sign of every component except w.

---

bool **is_equal_approx**(to: Quaternion) const 🔗

Returns `true` if this quaternion and `to` are approximately equal, by running @GlobalScope.is_equal_approx on each component.

---

bool **is_finite**() const 🔗

Returns `true` if this quaternion is finite, by calling @GlobalScope.is_finite on each component.

---

bool **is_normalized**() const 🔗

Returns `true` if this quaternion is normalized. See also normalized.

---

float **length**() const 🔗

Returns this quaternion's length, also called magnitude.

---

float **length_squared**() const 🔗

Returns this quaternion's length, squared.

Note: This method is faster than length, so prefer it if you only need to compare quaternion lengths.

Quaternion **log**() const 🔗

Returns the logarithm of this quaternion. Multiplies this quaternion's rotation axis by its rotation angle, and stores the result in the returned quaternion's vector part (x, y, and z). The returned quaternion's real part (w) is always `0.0`.

Quaternion **normalized**() const 🔗

Returns a copy of this quaternion, normalized so that its length is `1.0`. See also is_normalized.

Quaternion **slerp**(to: Quaternion, weight: float) const

Performs a spherical-linear interpolation with the `to` quaternion, given a `weight` and returns the result. Both this quaternion and `to` must be normalized.

Quaternion **slerpni**(to: Quaternion, weight: float) const

Performs a spherical-linear interpolation with the `to` quaternion, given a `weight` and returns the result. Unlike slerp, this method does not check if the rotation path is smaller than 90 degrees. Both this quaternion and `to` must be normalized.

Quaternion **spherical_cubic_interpolate**(b: Quaternion, pre_a: Quaternion, post_b: Quaternion, weight: float) const

Performs a spherical cubic interpolation between quaternions `pre_a`, this vector, `b`, and `post_b`, by the given amount `weight`.

Quaternion **spherical_cubic_interpolate_in_time**(b: Quaternion, pre_a: Quaternion, post_b: Quaternion, weight: float, b_t: float, pre_a_t: float, post_b_t: float) const

Performs a spherical cubic interpolation between quaternions `pre_a`, this vector, `b`, and `post_b`, by the given amount `weight`.

It can perform smoother interpolation than spherical_cubic_interpolate by the time values.

Operator Descriptions

bool **operator !=**(right: Quaternion)

Returns `true` if the components of both quaternions are not exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Quaternion **operator ***(right: Quaternion)

Composes (multiplies) two quaternions. This rotates the `right` quaternion (the child) by this quaternion (the parent).

Vector3 **operator ***(right: Vector3)

Rotates (multiplies) the `right` vector by this quaternion, returning a Vector3.

Quaternion **operator ***(right: float)

Multiplies each component of the **Quaternion** by the right float value.

This operation is not meaningful on its own, but it can be used as a part of a larger expression.

Quaternion **operator ***(right: int)

Multiplies each component of the **Quaternion** by the right int value.

This operation is not meaningful on its own, but it can be used as a part of a larger expression.

Quaternion **operator +**(right: Quaternion)

Adds each component of the left **Quaternion** to the right **Quaternion**.

This operation is not meaningful on its own, but it can be used as a part of a larger expression, such as approximating an intermediate rotation between two nearby rotations.

Quaternion **operator -**(right: Quaternion)

Subtracts each component of the left **Quaternion** by the right **Quaternion**.

This operation is not meaningful on its own, but it can be used as a part of a larger expression.

Quaternion **operator /**(right: float)

Divides each component of the **Quaternion** by the right float value.

This operation is not meaningful on its own, but it can be used as a part of a larger expression.

Quaternion **operator /**(right: int)

Divides each component of the **Quaternion** by the right int value.

This operation is not meaningful on its own, but it can be used as a part of a larger expression.

bool **operator ==**(right: Quaternion)

Returns `true` if the components of both quaternions are exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

---

float **operator [ ]**(index: int)

Accesses each component of this quaternion by their index.

Index `0` is the same as x, index `1` is the same as y, index `2` is the same as z, and index `3` is the same as w.

Quaternion **operator unary+**()

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

Quaternion **operator unary-**()

Returns the negative value of the **Quaternion**. This is the same as multiplying all components by `-1`. This operation results in a quaternion that represents the same rotation.

Rect2

A 2D axis-aligned bounding box using floating-point coordinates.

Description

The **Rect2** built-in Variant type represents an axis-aligned rectangle in a 2D space. It is defined by its position and size, which are Vector2. It is frequently used for fast overlap tests (see intersects). Although **Rect2** itself is axis-aligned, it can be combined with Transform2D to represent a rotated or skewed rectangle.

For integer coordinates, use Rect2i. The 3D equivalent to **Rect2** is AABB.

Note: Negative values for size are not supported. With negative size, most **Rect2** methods do not work correctly. Use abs to get an equivalent **Rect2** with a non-negative size.

Note: In a boolean context, a **Rect2** evaluates to `false` if both position and size are zero (equal to Vector2.ZERO). Otherwise, it always evaluates to `true`.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Math documentation index

- Vector math
- Advanced vector math

Properties

Vector2	end	`Vector2(0, 0)`
Vector2	position	`Vector2(0, 0)`
Vector2	size	`Vector2(0, 0)`

Constructors

Rect2	Rect2()
Rect2	Rect2(from: Rect2)
Rect2	Rect2(from: Rect2i)
Rect2	Rect2(position: Vector2, size: Vector2)
Rect2	Rect2(x: float, y: float, width: float, height: float)

Methods

Rect2	abs() const
bool	encloses(b: Rect2) const
Rect2	expand(to: Vector2) const
float	get_area() const
Vector2	get_center() const
Rect2	grow(amount: float) const
Rect2	grow_individual(left: float, top: float, right: float, bottom: float) const
Rect2	grow_side(side: int, amount: float) const
bool	has_area() const
bool	has_point(point: Vector2) const
Rect2	intersection(b: Rect2) const
bool	intersects(b: Rect2, include_borders: bool = false) const

bool	is_equal_approx(rect: Rect2) const
bool	is_finite() const
Rect2	merge(b: Rect2) const

Operators

bool	operator !=(right: Rect2)
Rect2	operator *(right: Transform2D)
bool	operator ==(right: Rect2)

Property Descriptions

Vector2 **end** = `Vector2(0, 0)` 🔗

The ending point. This is usually the bottom-right corner of the rectangle, and is equivalent to `position + size`. Setting this point affects the size.

Vector2 **position** = `Vector2(0, 0)` 🔗

The origin point. This is usually the top-left corner of the rectangle.

Vector2 **size** = `Vector2(0, 0)` 🔗

The rectangle's width and height, starting from position. Setting this value also affects the end point.

Note: It's recommended setting the width and height to non-negative values, as most methods in Godot assume that the position is the top-left corner, and the end is the bottom-right corner. To get an equivalent rectangle with non-negative size, use abs.

Constructor Descriptions

Rect2 **Rect2**() 🔗

Constructs a **Rect2** with its position and size set to Vector2.ZERO.

Rect2 **Rect2**(from: Rect2)

Constructs a **Rect2** as a copy of the given **Rect2**.

Rect2 **Rect2**(from: Rect2i)

Constructs a **Rect2** from a Rect2i.

Rect2 **Rect2**(position: Vector2, size: Vector2)

Constructs a **Rect2** by `position` and `size`.

Rect2 **Rect2**(x: float, y: float, width: float, height: float)

Constructs a **Rect2** by setting its position to (`x`, `y`), and its size to (`width`, `height`).

Method Descriptions

Rect2 **abs**() const 🔗

Returns a **Rect2** equivalent to this rectangle, with its width and height modified to be non-negative values, and with its position being the top-left corner of the rectangle.

GDScript

```
var rect = Rect2(25, 25, -100, -50)
var absolute = rect.abs() # absolute is Rect2(-75, -25, 100, 50)
```

C#

```
var rect = new Rect2(25, 25, -100, -50);
var absolute = rect.Abs(); // absolute is Rect2(-75, -25, 100, 50)
```

Note: It's recommended to use this method when size is negative, as most other methods in Godot assume that the position is the top-left corner, and the end is the bottom-right corner.

---

bool **encloses**(b: Rect2) const 🔗

Returns `true` if this rectangle *completely* encloses the `b` rectangle.

Rect2 **expand**(to: Vector2) const 🔗

Returns a copy of this rectangle expanded to align the edges with the given `to` point, if necessary.

GDScript

```
var rect = Rect2(0, 0, 5, 2)

rect = rect.expand(Vector2(10, 0)) # rect is Rect2(0, 0, 10, 2)
rect = rect.expand(Vector2(-5, 5)) # rect is Rect2(-5, 0, 15, 5)
```

C#

```
var rect = new Rect2(0, 0, 5, 2);

rect = rect.Expand(new Vector2(10, 0)); // rect is Rect2(0, 0, 10, 2)
rect = rect.Expand(new Vector2(-5, 5)); // rect is Rect2(-5, 0, 15, 5)
```

---

float **get_area**() const

Returns the rectangle's area. This is equivalent to `size.x * size.y`. See also has_area.

Vector2 **get_center**() const

Returns the center point of the rectangle. This is the same as `position + (size / 2.0)`.

Rect2 **grow**(amount: float) const

Returns a copy of this rectangle extended on all sides by the given `amount`. A negative `amount` shrinks the rectangle instead. See also grow_individual and grow_side.

GDScript

```gdscript
var a = Rect2(4, 4, 8, 8).grow(4) # a is Rect2(0, 0, 16, 16)
var b = Rect2(0, 0, 8, 4).grow(2) # b is Rect2(-2, -2, 12, 8)
```

C#

```csharp
var a = new Rect2(4, 4, 8, 8).Grow(4); // a is Rect2(0, 0, 16, 16)
var b = new Rect2(0, 0, 8, 4).Grow(2); // b is Rect2(-2, -2, 12, 8)
```

Rect2 **grow_individual**(left: float, top: float, right: float, bottom: float) const

Returns a copy of this rectangle with its `left`, `top`, `right`, and `bottom` sides extended by the given amounts. Negative values shrink the sides, instead. See also grow and grow_side.

Rect2 **grow_side**(side: int, amount: float) const

Returns a copy of this rectangle with its `side` extended by the given `amount` (see Side constants). A negative `amount` shrinks the rectangle, instead. See also grow and grow_individual.

bool **has_area**() const

Returns `true` if this rectangle has positive width and height. See also get_area.

bool **has_point**(point: Vector2) const

Returns `true` if the rectangle contains the given `point`. By convention, points on the right and bottom edges are **not** included.

Note: This method is not reliable for **Rect2** with a *negative* size. Use abs first to get a valid rectangle.

Rect2 **intersection**(b: Rect2) const 🔗

Returns the intersection between this rectangle and `b`. If the rectangles do not intersect, returns an empty **Rect2**.

GDScript

```
var rect1 = Rect2(0, 0, 5, 10)
var rect2 = Rect2(2, 0, 8, 4)

var a = rect1.intersection(rect2) # a is Rect2(2, 0, 3, 4)
```

C#

```
var rect1 = new Rect2(0, 0, 5, 10);
var rect2 = new Rect2(2, 0, 8, 4);

var a = rect1.Intersection(rect2); // a is Rect2(2, 0, 3, 4)
```

Note: If you only need to know whether two rectangles are overlapping, use intersects, instead.

---

bool **intersects**(b: Rect2, include_borders: bool = false) const 🔗

Returns `true` if this rectangle overlaps with the `b` rectangle. The edges of both rectangles are excluded, unless `include_borders` is `true`.

---

bool **is_equal_approx**(rect: Rect2) const 🔗

Returns `true` if this rectangle and `rect` are approximately equal, by calling Vector2.is_equal_approx on the position and the size.

---

bool **is_finite**() const

Returns `true` if this rectangle's values are finite, by calling Vector2.is_finite on the position and the size.

Rect2 **merge**(b: Rect2) const

Returns a **Rect2** that encloses both this rectangle and `b` around the edges. See also encloses.

Operator Descriptions

bool **operator !=**(right: Rect2)

Returns `true` if the position or size of both rectangles are not equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Rect2 **operator ***(right: Transform2D)

Inversely transforms (multiplies) the **Rect2** by the given Transform2D transformation matrix, under the assumption that the transformation basis is orthonormal (i.e. rotation/reflection is fine, scaling/skew is not).

`rect * transform` is equivalent to `transform.inverse() * rect`. See Transform2D.inverse.

For transforming by inverse of an affine transformation (e.g. with scaling) `transform.affine_inverse() * rect` can be used instead. See Transform2D.affine_inverse.

---

bool **operator ==**(right: Rect2) 🔗

Returns `true` if both position and size of the rectangles are exactly equal, respectively.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Rect2i

A 2D axis-aligned bounding box using integer coordinates.

Description

The **Rect2i** built-in Variant type represents an axis-aligned rectangle in a 2D space, using integer coordinates. It is defined by its position and size, which are Vector2i. Because it does not rotate, it is frequently used for fast overlap tests (see intersects).

For floating-point coordinates, see Rect2.

Note: Negative values for size are not supported. With negative size, most **Rect2i** methods do not work correctly. Use abs to get an equivalent **Rect2i** with a non-negative size.

Note: In a boolean context, a **Rect2i** evaluates to `false` if both position and size are zero (equal to Vector2i.ZERO). Otherwise, it always evaluates to `true`.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Math documentation index
- Vector math

Properties

Vector2i	end	`Vector2i(0, 0)`
Vector2i	position	`Vector2i(0, 0)`
Vector2i	size	`Vector2i(0, 0)`

Constructors

Rect2i	Rect2i()
Rect2i	Rect2i(from: Rect2i)
Rect2i	Rect2i(from: Rect2)
Rect2i	Rect2i(position: Vector2i, size: Vector2i)
Rect2i	Rect2i(x: int, y: int, width: int, height: int)

Methods

Rect2i	abs() const

bool	encloses(b: Rect2i) const
Rect2i	expand(to: Vector2i) const
int	get_area() const
Vector2i	get_center() const
Rect2i	grow(amount: int) const
Rect2i	grow_individual(left: int, top: int, right: int, bottom: int) const
Rect2i	grow_side(side: int, amount: int) const
bool	has_area() const
bool	has_point(point: Vector2i) const
Rect2i	intersection(b: Rect2i) const
bool	intersects(b: Rect2i) const
Rect2i	merge(b: Rect2i) const

Operators

bool operator !=(right: Rect2i)

bool operator ==(right: Rect2i)

Property Descriptions

Vector2i **end** = `Vector2i(0, 0)`

The ending point. This is usually the bottom-right corner of the rectangle, and is equivalent to `position + size`. Setting this point affects the size.

Vector2i **position** = `Vector2i(0, 0)`

The origin point. This is usually the top-left corner of the rectangle.

Vector2i **size** = `Vector2i(0, 0)`

The rectangle's width and height, starting from position. Setting this value also affects the end point.

Note: It's recommended setting the width and height to non-negative values, as most methods in Godot assume that the position is the top-left corner, and the end is the bottom-right corner. To get an equivalent rectangle with non-negative size, use abs.

Constructor Descriptions

Rect2i **Rect2i**() 🔗

Constructs a **Rect2i** with its position and size set to Vector2i.ZERO.

Rect2i **Rect2i**(from: Rect2i)

Constructs a **Rect2i** as a copy of the given **Rect2i**.

Rect2i **Rect2i**(from: Rect2)

Constructs a **Rect2i** from a Rect2. The floating-point coordinates are truncated.

Rect2i **Rect2i**(position: Vector2i, size: Vector2i)

Constructs a **Rect2i** by `position` and `size`.

Rect2i **Rect2i**(x: int, y: int, width: int, height: int)

Constructs a **Rect2i** by setting its position to (`x`, `y`), and its size to (`width`, `height`).

Method Descriptions

Rect2i **abs**() const 🔗

Returns a **Rect2i** equivalent to this rectangle, with its width and height modified to be non-negative values, and with its position being the top-left corner of the rectangle.

GDScript

```
var rect = Rect2i(25, 25, -100, -50)
var absolute = rect.abs() # absolute is Rect2i(-75, -25, 100,
50)
```

C#

```
var rect = new Rect2I(25, 25, -100, -50);
var absolute = rect.Abs(); // absolute is Rect2I(-75, -25, 100,
50)
```

Note: It's recommended to use this method when size is negative, as most other methods in Godot assume that the position is the top-left corner, and the end is the bottom-right corner.

---

bool **encloses**(b: Rect2i) const

Returns `true` if this **Rect2i** completely encloses another one.

Rect2i **expand**(to: Vector2i) const

Returns a copy of this rectangle expanded to align the edges with the given `to` point, if necessary.

GDScript

```
var rect = Rect2i(0, 0, 5, 2)

rect = rect.expand(Vector2i(10, 0)) # rect is Rect2i(0, 0, 10,
2)
rect = rect.expand(Vector2i(-5, 5)) # rect is Rect2i(-5, 0, 15,
5)
```

C#

```
var rect = new Rect2I(0, 0, 5, 2);

rect = rect.Expand(new Vector2I(10, 0)); // rect is Rect2I(0, 0, 10, 2)
rect = rect.Expand(new Vector2I(-5, 5)); // rect is Rect2I(-5, 0, 15, 5)
```

---

int **get_area**() const

Returns the rectangle's area. This is equivalent to `size.x * size.y`. See also has_area.

Vector2i **get_center**() const

Returns the center point of the rectangle. This is the same as `position + (size / 2)`.

Note: If the size is odd, the result will be rounded towards position.

Rect2i **grow**(amount: int) const

Returns a copy of this rectangle extended on all sides by the given `amount`. A negative `amount` shrinks the rectangle instead. See also grow_individual and grow_side.

GDScript

```
var a = Rect2i(4, 4, 8, 8).grow(4) # a is Rect2i(0, 0, 16, 16)
var b = Rect2i(0, 0, 8, 4).grow(2) # b is Rect2i(-2, -2, 12, 8)
```

C#

```
var a = new Rect2I(4, 4, 8, 8).Grow(4); // a is Rect2I(0, 0, 16, 16)
var b = new Rect2I(0, 0, 8, 4).Grow(2); // b is Rect2I(-2, -2, 12, 8)
```

Rect2i **grow_individual**(left: int, top: int, right: int, bottom: int) const

Returns a copy of this rectangle with its `left`, `top`, `right`, and `bottom` sides extended by the given amounts. Negative values shrink the sides, instead. See also grow and grow_side.

Rect2i **grow_side**(side: int, amount: int) const

Returns a copy of this rectangle with its `side` extended by the given `amount` (see Side constants). A negative `amount` shrinks the rectangle, instead. See also grow and grow_individual.

---

bool **has_area**() const

Returns `true` if this rectangle has positive width and height. See also get_area.

---

bool **has_point**(point: Vector2i) const

Returns `true` if the rectangle contains the given `point`. By convention, points on the right and bottom edges are **not** included.

Note: This method is not reliable for **Rect2i** with a *negative* size. Use abs first to get a valid rectangle.

Rect2i **intersection**(b: Rect2i) const

Returns the intersection between this rectangle and `b`. If the rectangles do not intersect, returns an empty **Rect2i**.

GDScript

```
var a = Rect2i(0, 0, 5, 10)
var b = Rect2i(2, 0, 8, 4)

var c = a.intersection(b) # c is Rect2i(2, 0, 3, 4)
```

C#

```
var a = new Rect2I(0, 0, 5, 10);
var b = new Rect2I(2, 0, 8, 4);

var c = rect1.Intersection(rect2); // c is Rect2I(2, 0, 3, 4)
```

Note: If you only need to know whether two rectangles are overlapping, use intersects, instead.

---

bool **intersects**(b: Rect2i) const 🔗

Returns `true` if this rectangle overlaps with the `b` rectangle. The edges of both rectangles are excluded.

Rect2i **merge**(b: Rect2i) const 🔗

Returns a **Rect2i** that encloses both this rectangle and `b` around the edges. See also encloses.

Operator Descriptions

bool **operator !=**(right: Rect2i) 🔗

Returns `true` if the position or size of both rectangles are not equal.

---

bool **operator ==**(right: Rect2i) 🔗

Returns `true` if both position and size of the rectangles are equal, respectively.

RID

A handle for a Resource's unique identifier.

Description

The RID Variant type is used to access a low-level resource by its unique ID. RIDs are opaque, which means they do not grant access to the resource by themselves. They are used by the low-level server classes, such as DisplayServer, RenderingServer, TextServer, etc.

A low-level resource may correspond to a high-level Resource, such as Texture or Mesh.

Note: RIDs are only useful during the current session. It won't correspond to a similar resource if sent over a network, or loaded from a file at a later time.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

RID RID()

RID RID(from: RID)

Methods

int	get_id() const
bool	is_valid() const

Operators

bool	operator !=(right: RID)
bool	operator <(right: RID)
bool	operator <=(right: RID)
bool	operator ==(right: RID)
bool	operator >(right: RID)
bool	operator >=(right: RID)

Constructor Descriptions

RID **RID**() 🔗

Constructs an empty **RID** with the invalid ID `0`.

RID **RID**(from: RID)

Constructs a **RID** as a copy of the given **RID**.

Method Descriptions

int **get_id**() const

Returns the ID of the referenced low-level resource.

bool **is_valid**() const

Returns `true` if the **RID** is not `0`.

Operator Descriptions

bool **operator !=**(right: RID)

Returns `true` if the **RID**s are not equal.

bool **operator <**(right: RID)

Returns `true` if the **RID**'s ID is less than `right`'s ID.

bool **operator <=**(right: RID)

Returns `true` if the **RID**'s ID is less than or equal to `right`'s ID.

---

bool **operator ==**(right: RID) 🔗

Returns `true` if both **RID**s are equal, which means they both refer to the same low-level resource.

---

bool **operator >**(right: RID) 🔗

Returns `true` if the **RID**'s ID is greater than `right`'s ID.

---

bool **operator >=**(right: RID) 🔗

Returns `true` if the **RID**'s ID is greater than or equal to `right`'s ID.

Signal

A built-in type representing a signal of an Object.

Description

Signal is a built-in Variant type that represents a signal of an Object instance. Like all Variant types, it can be stored in variables and passed to functions. Signals allow all connected Callables (and by extension their respective objects) to listen and react to events, without directly referencing one another. This keeps the code flexible and easier to manage.

In GDScript, signals can be declared with the `signal` keyword. In C#, you may use the `[Signal]` attribute on a delegate.

GDScript

```
signal attacked

# Additional arguments may be declared.
# These arguments must be passed when the signal is emitted.
signal item_dropped(item_name, amount)
```

C#

```
[Signal]
delegate void AttackedEventHandler();

// Additional arguments may be declared.
// These arguments must be passed when the signal is emitted.
[Signal]
delegate void ItemDroppedEventHandler(string itemName, int
amount);
```

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Using Signals
- GDScript Basics

Constructors

Signal	Signal()
Signal	Signal(from: Signal)
Signal	Signal(object: Object, signal: StringName)

Methods

int	connect(callable: Callable, flags: int = 0)
void	disconnect(callable: Callable)
void	emit(...) vararg const

Array	get_connections() const
StringName	get_name() const
Object	get_object() const
int	get_object_id() const
bool	is_connected(callable: Callable) const
bool	is_null() const

Operators

bool	operator !=(right: Signal)
bool	operator ==(right: Signal)

Constructor Descriptions

Signal **Signal**() 🔗

Constructs an empty **Signal** with no object nor signal name bound.

Signal **Signal**(from: Signal)

Constructs a **Signal** as a copy of the given **Signal**.

Signal **Signal**(object: Object, signal: StringName)

Creates a new **Signal** named `signal` in the specified `object`.

Method Descriptions

int **connect**(callable: Callable, flags: int = 0) 🔗

Connects this signal to the specified `callable`. Optional `flags` can be also added to configure the connection's behavior (see ConnectFlags constants). You can provide additional arguments to the connected `callable` by using Callable.bind.

A signal can only be connected once to the same Callable. If the signal is already connected, returns @GlobalScope.ERR_INVALID_PARAMETER and pushes an error message, unless the signal is connected with Object.CONNECT_REFERENCE_COUNTED. To prevent this, use is_connected first to check for existing connections.

```
for button in $Buttons.get_children():
    button.pressed.connect(_on_pressed.bind(button))

func _on_pressed(button):
    print(button.name, " was pressed")
```

---

void **disconnect**(callable: Callable) 🔗

Disconnects this signal from the specified Callable. If the connection does not exist, generates an error. Use is_connected to make sure that the connection exists.

---

void **emit**(...) vararg const

Emits this signal. All Callables connected to this signal will be triggered. This method supports a variable number of arguments, so parameters can be passed as a comma separated list.

Array **get_connections**() const

Returns an Array of connections for this signal. Each connection is represented as a Dictionary that contains three entries:

- `signal` is a reference to this signal;
- `callable` is a reference to the connected Callable;
- `flags` is a combination of ConnectFlags.

StringName **get_name**() const

Returns the name of this signal.

Object **get_object**() const

Returns the object emitting this signal.

---

int **get_object_id**() const

Returns the ID of the object emitting this signal (see Object.get_instance_id).

---

bool **is_connected**(callable: Callable) const

Returns `true` if the specified Callable is connected to this signal.

---

bool **is_null**() const

Returns `true` if the signal's name does not exist in its object, or the object is not valid.

Operator Descriptions

bool **operator !=**(right: Signal)

Returns `true` if the signals do not share the same object and name.

---

bool **operator ==**(right: Signal)

Returns `true` if both signals share the same object and name.

String

A built-in type for strings.

Description

This is the built-in string Variant type (and the one used by GDScript). Strings may contain any number of Unicode characters, and expose methods useful for manipulating and generating strings. Strings are reference-counted and use a copy-on-write approach (every modification to a string returns a new **String**), so passing them around is cheap in resources.

Some string methods have corresponding variations. Variations suffixed with `n` (countn, findn, replacen, etc.) are **case-insensitive** (they make no distinction between uppercase and lowercase letters). Method variations prefixed with `r` (rfind, rsplit, etc.) are reversed, and start from the end of the string, instead of the beginning.

Note: In a boolean context, a string will evaluate to `false` if it is empty (`""`). Otherwise, a string will always evaluate to `true`. The `not` operator cannot be used. Instead, is_empty should be used to check for empty strings.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- GDScript format strings

Constructors

String	String()
String	String(from: String)
String	String(from: NodePath)
String	String(from: StringName)

Methods

bool	begins_with(text: String) const
PackedStringArray	bigrams() const
int	bin_to_int() const
String	c_escape() const
String	c_unescape() const
String	capitalize() const

int	casecmp_to(to: String) const
String	chr(char: int) static
bool	contains(what: String) const
bool	containsn(what: String) const
int	count(what: String, from: int = 0, to: int = 0) const
int	countn(what: String, from: int = 0, to: int = 0) const
String	dedent() const
bool	ends_with(text: String) const
String	erase(position: int, chars: int = 1) const
int	filecasecmp_to(to: String) const
int	filenocasecmp_to(to: String) const
int	find(what: String, from: int = 0) const

int	findn(what: String, from: int = 0) const
String	format(values: Variant, placeholder: String = "{_}") const
String	get_base_dir() const
String	get_basename() const
String	get_extension() const
String	get_file() const
String	get_slice(delimiter: String, slice: int) const
int	get_slice_count(delimiter: String) const
String	get_slicec(delimiter: int, slice: int) const
int	hash() const
PackedByteArray	hex_decode() const
int	hex_to_int() const

String	humanize_size(size: int) static
String	indent(prefix: String) const
String	insert(position: int, what: String) const
bool	is_absolute_path() const
bool	is_empty() const
bool	is_relative_path() const
bool	is_subsequence_of(text: String) const
bool	is_subsequence_ofn(text: String) const
bool	is_valid_filename() const
bool	is_valid_float() const
bool	is_valid_hex_number(with_prefix: bool = false) const
bool	is_valid_html_color() const

bool	is_valid_identifier() const
bool	is_valid_int() const
bool	is_valid_ip_address() const
String	join(parts: PackedStringArray) const
String	json_escape() const
String	left(length: int) const
int	length() const
String	lpad(min_length: int, character: String = " ") const
String	lstrip(chars: String) const
bool	match(expr: String) const
bool	matchn(expr: String) const
PackedByteArray	md5_buffer() const

String	md5_text() const
int	naturalcasecmp_to(to: String) const
int	naturalnocasecmp_to(to: String) const
int	nocasecmp_to(to: String) const
String	num(number: float, decimals: int = -1) static
String	num_int64(number: int, base: int = 10, capitalize_hex: bool = false) static
String	num_scientific(number: float) static
String	num_uint64(number: int, base: int = 10, capitalize_hex: bool = false) static
String	pad_decimals(digits: int) const
String	pad_zeros(digits: int) const
String	path_join(file: String) const
String	repeat(count: int) const

String	replace(what: String, forwhat: String) const
String	replacen(what: String, forwhat: String) const
String	reverse() const
int	rfind(what: String, from: int = -1) const
int	rfindn(what: String, from: int = -1) const
String	right(length: int) const
String	rpad(min_length: int, character: String = " ") const
PackedStringArray	rsplit(delimiter: String = "", allow_empty: bool = true, maxsplit: int = 0) const
String	rstrip(chars: String) const
PackedByteArray	sha1_buffer() const
String	sha1_text() const
PackedByteArray	sha256_buffer() const

String	sha256_text() const
float	similarity(text: String) const
String	simplify_path() const
PackedStringArray	split(delimiter: String = "", allow_empty: bool = true, maxsplit: int = 0) const
PackedFloat64Array	split_floats(delimiter: String, allow_empty: bool = true) const
String	strip_edges(left: bool = true, right: bool = true) const
String	strip_escapes() const
String	substr(from: int, len: int = -1) const
PackedByteArray	to_ascii_buffer() const
String	to_camel_case() const
float	to_float() const
int	to_int() const

String	to_lower() const
String	to_pascal_case() const
String	to_snake_case() const
String	to_upper() const
PackedByteArray	to_utf8_buffer() const
PackedByteArray	to_utf16_buffer() const
PackedByteArray	to_utf32_buffer() const
PackedByteArray	to_wchar_buffer() const
String	trim_prefix(prefix: String) const
String	trim_suffix(suffix: String) const
int	unicode_at(at: int) const
String	uri_decode() const
String	uri_encode() const

String	validate_filename() const
String	validate_node_name() const
String	xml_escape(escape_quotes: bool = false) const
String	xml_unescape() const

Operators

bool	operator !=(right: String)
bool	operator !=(right: StringName)
String	operator %(right: Variant)
String	operator +(right: String)
String	operator +(right: StringName)
bool	operator <(right: String)
bool	operator <=(right: String)

bool	operator ==(right: String)
bool	operator ==(right: StringName)
bool	operator >(right: String)
bool	operator >=(right: String)
String	operator [](index: int)

Constructor Descriptions

String **String**() 🔗

Constructs an empty **String** (`""`).

String **String**(from: String)

Constructs a **String** as a copy of the given **String**.

String **String**(from: NodePath)

Constructs a new **String** from the given NodePath.

String **String**(from: StringName)

Constructs a new **String** from the given StringName.

Method Descriptions

bool **begins_with**(text: String) const

Returns `true` if the string begins with the given `text`. See also ends_with.

PackedStringArray **bigrams**() const

Returns an array containing the bigrams (pairs of consecutive characters) of this string.

```
print("Get up!".bigrams()) # Prints ["Ge", "et", "t ", " u", "up", "p!"]
```

---

int **bin_to_int**() const

Converts the string representing a binary number into an int. The string may optionally be prefixed with `"0b"`, and an additional `-` prefix for negative numbers.

GDScript

```
print("101".bin_to_int())   # Prints 5
print("0b101".bin_to_int()) # Prints 5
print("-0b10".bin_to_int()) # Prints -2
```

C#

```
GD.Print("101".BinToInt());   // Prints 5
GD.Print("0b101".BinToInt()); // Prints 5
GD.Print("-0b10".BinToInt()); // Prints -2
```

String **c_escape**() const

Returns a copy of the string with special characters escaped using the C language standard.

String **c_unescape**() const

Returns a copy of the string with escaped characters replaced by their meanings. Supported escape sequences are `\'`, `\"`, `\\`, `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v`.

Note: Unlike the GDScript parser, this method doesn't support the `\uXXXX` escape sequence.

String **capitalize**() const

Changes the appearance of the string: replaces underscores (_) with spaces, adds spaces before uppercase letters in the middle of a word, converts all letters to lowercase, then converts the first one and each one following a space to uppercase.

GDScript

```
"move_local_x".capitalize()   # Returns "Move Local X"
"sceneFile_path".capitalize() # Returns "Scene File Path"
"2D, FPS, PNG".capitalize()   # Returns "2d, Fps, Png"
```

C#

```
"move_local_x".Capitalize();   // Returns "Move Local X"
"sceneFile_path".Capitalize(); // Returns "Scene File Path"
"2D, FPS, PNG".Capitalize();   // Returns "2d, Fps, Png"
```

---

int **casecmp_to**(to: String) const

Performs a case-sensitive comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "Less than" and "greater than" are determined by the Unicode code points [https://en.wikipedia.org/wiki/List_of_Unicode_characters] of each string, which roughly matches the alphabetical order.

With different string lengths, returns `1` if this string is longer than the `to` string, or `-1` if shorter. Note that the length of empty strings is *always* `0`.

To get a bool result from a string comparison, use the `==` operator instead. See also nocasecmp_to, filecasecmp_to, and naturalcasecmp_to.

String **chr**(char: int) static 🔗

Returns a single Unicode character from the decimal `char`. You may use unicodelookup.com [https://unicodelookup.com/] or unicode.org [https://www.unicode.org/charts/] as points of reference.

```
print(String.chr(65))      # Prints "A"
print(String.chr(129302)) # Prints "🤖" (robot face emoji)
```

---

bool **contains**(what: String) const 🔗

Returns `true` if the string contains `what`. In GDScript, this corresponds to the `in` operator.

GDScript

```
print("Node".contains("de")) # Prints true
print("team".contains("I"))  # Prints false
print("I" in "team")         # Prints false
```

C#

```
GD.Print("Node".Contains("de")); // Prints true
GD.Print("team".Contains("I"));  // Prints false
```

If you need to know where `what` is within the string, use find. See also containsn.

---

bool **containsn**(what: String) const

Returns `true` if the string contains `what`, **ignoring case**.

If you need to know where `what` is within the string, use findn. See also contains.

---

int **count**(what: String, from: int = 0, to: int = 0) const

Returns the number of occurrences of the substring `what` between `from` and `to` positions. If `to` is 0, the search continues until the end of the string.

---

int **countn**(what: String, from: int = 0, to: int = 0) const

Returns the number of occurrences of the substring `what` between `from` and `to` positions, **ignoring case**. If `to` is 0, the search continues until the end of the string.

String **dedent**() const

Returns a copy of the string with indentation (leading tabs and spaces) removed. See also indent to add indentation.

---

bool **ends_with**(text: String) const

Returns `true` if the string ends with the given `text`. See also begins_with.

String **erase**(position: int, chars: int = 1) const 🔗

Returns a string with `chars` characters erased starting from `position`. If `chars` goes beyond the string's length given the specified `position`, fewer characters will be erased from the returned string. Returns an empty string if either `position` or `chars` is negative. Returns the original string unmodified if `chars` is `0`.

---

int **filecasecmp_to**(to: String) const 🔗

Like naturalcasecmp_to but prioritizes strings that begin with periods (`.`) and underscores (`_`) before any other character. Useful when sorting folders or file names.

To get a bool result from a string comparison, use the `==` operator instead. See also filenocasecmp_to, naturalcasecmp_to, and casecmp_to.

---

int **filenocasecmp_to**(to: String) const 🔗

Like naturalnocasecmp_to but prioritizes strings that begin with periods (`.`) and underscores (`_`) before any other character. Useful when sorting folders or file names.

To get a bool result from a string comparison, use the `==` operator instead. See also filecasecmp_to, naturalnocasecmp_to, and nocasecmp_to.

---

int **find**(what: String, from: int = 0) const 🔗

Returns the index of the **first** occurrence of `what` in this string, or `-1` if there are none. The search's start can be specified with `from`, continuing to the end of the string.

GDScript

```
print("Team".find("I")) # Prints -1

print("Potato".find("t"))    # Prints 2
print("Potato".find("t", 3)) # Prints 4
print("Potato".find("t", 5)) # Prints -1
```

C#

```
GD.Print("Team".Find("I")); // Prints -1

GD.Print("Potato".Find("t"));    // Prints 2
GD.Print("Potato".Find("t", 3)); // Prints 4
GD.Print("Potato".Find("t", 5)); // Prints -1
```

Note: If you just want to know whether the string contains `what`, use contains. In GDScript, you may also use the `in` operator.

---

int **findn**(what: String, from: int = 0) const 🔗

Returns the index of the **first case-insensitive** occurrence of `what` in this string, or `-1` if there are none. The starting search index can be specified with `from`, continuing to the end of the string.

String **format**(values: Variant, placeholder: String = "{_}") const 🔗

Formats the string by replacing all occurrences of `placeholder` with the elements of `values`.

`values` can be a Dictionary or an Array. Any underscores in `placeholder` will be replaced with the corresponding keys in advance.

Array elements use their index as keys.

```
# Prints "Waiting for Godot is a play by Samuel Beckett, and
Godot Engine is named after it."
var use_array_values = "Waiting for {0} is a play by {1}, and
{0} Engine is named after it."
print(use_array_values.format(["Godot", "Samuel Beckett"]))

# Prints "User 42 is Godot."
print("User {id} is {name}.".format({"id": 42, "name":
"Godot"}))
```

Some additional handling is performed when `values` is an [Array](). If `placeholder` does not contain an underscore, the elements of the `values` array will be used to replace one occurrence of the placeholder in order; If an element of `values` is another 2-element array, it'll be interpreted as a key-value pair.

```
# Prints "User 42 is Godot."
print("User {} is {}.".format([42, "Godot"], "{}"))
print("User {id} is {name}.".format([["id", 42], ["name",
"Godot"]]))
```

See also the [GDScript format string]() tutorial.

Note: The replacement of placeholders is not done all at once, instead each placeholder is replaced in the order they are passed, this means that if one of the replacement strings contains a key it will also be replaced. This can be very powerful, but can also cause unexpected results if you are not careful. If you do not need to perform replacement in the replacement strings, make sure your replacements do not contain placeholders to ensure reliable results.

```
print("{0} {1}".format(["{1}", "x"]))                       #
Prints "x x".
print("{0} {1}".format(["x", "{0}"]))                       #
Prints "x {0}".
print("{foo} {bar}".format({"foo": "{bar}", "bar": "baz"})) #
Prints "baz baz".
```

```
print("{foo} {bar}".format({"bar": "baz", "foo": "{bar}"})) #
Prints "{bar} baz".
```

Note: In C#, it's recommended to interpolate strings with "$" [https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated], instead.

String **get_base_dir**() const

If the string is a valid file path, returns the base directory name.

```
var dir_path = "/path/to/file.txt".get_base_dir() # dir_path is
"/path/to"
```

String **get_basename**() const

If the string is a valid file path, returns the full file path, without the extension.

```
var base = "/path/to/file.txt".get_basename() # base is
"/path/to/file"
```

String **get_extension**() const

If the string is a valid file name or path, returns the file extension without the leading period (`.`). Otherwise, returns an empty string.

```
var a = "/path/to/file.txt".get_extension() # a is "txt"
var b = "cool.txt".get_extension()          # b is "txt"
var c = "cool.font.tres".get_extension()    # c is "tres"
var d = ".pack1".get_extension()            # d is "pack1"

var e = "file.txt.".get_extension()  # e is ""
var f = "file.txt..".get_extension() # f is ""
var g = "txt".get_extension()         # g is ""
var h = "".get_extension()            # h is ""
```

String **get_file**() const

If the string is a valid file path, returns the file name, including the extension.

```
var file = "/path/to/icon.png".get_file() # file is "icon.png"
```

String **get_slice**(delimiter: String, slice: int) const

Splits the string using a `delimiter` and returns the substring at index `slice`. Returns the original string if `delimiter` does not occur in the string. Returns an empty string if the `slice` does not exist.

This is faster than split, if you only need one substring.

Example:

```
print("i/am/example/hi".get_slice("/", 2)) # Prints "example"
```

---

int **get_slice_count**(delimiter: String) const

Returns the total number of slices when the string is split with the given `delimiter` (see split).

String **get_slicec**(delimiter: int, slice: int) const

Splits the string using a Unicode character with code `delimiter` and returns the substring at index `slice`. Returns an empty string if the `slice` does not exist.

This is faster than split, if you only need one substring.

---

int **hash**() const

Returns the 32-bit hash value representing the string's contents.

Note: Strings with equal hash values are *not* guaranteed to be the same, as a result of hash collisions. On the contrary, strings with different hash values are guaranteed to be different.

PackedByteArray **hex_decode**() const

Decodes a hexadecimal string as a PackedByteArray.

GDScript

```
var text = "hello world"
var encoded = text.to_utf8_buffer().hex_encode() # outputs "68656c6c6f20776f726c64"
print(buf.hex_decode().get_string_from_utf8())
```

C#

```
var text = "hello world";
var encoded = text.ToUtf8Buffer().HexEncode(); // outputs "68656c6c6f20776f726c64"
GD.Print(buf.HexDecode().GetStringFromUtf8());
```

---

int **hex_to_int**() const

Converts the string representing a hexadecimal number into an int. The string may be optionally prefixed with `"0x"`, and an additional `-` prefix for negative numbers.

GDScript

```
print("0xff".hex_to_int()) # Prints 255
print("ab".hex_to_int())    # Prints 171
```

C#

```
GD.Print("0xff".HexToInt()); // Prints 255
GD.Print("ab".HexToInt());   // Prints 171
```

String **humanize_size**(size: int) static

Converts `size` which represents a number of bytes into a human-readable form.

The result is in IEC prefix format [https://en.wikipedia.org/wiki/Binary_prefix#IEC_prefixes], which may end in either `"B"`, `"KiB"`, `"MiB"`, `"GiB"`, `"TiB"`, `"PiB"`, or `"EiB"`.

String **indent**(prefix: String) const

Indents every line of the string with the given `prefix`. Empty lines are not indented. See also dedent to remove indentation.

For example, the string can be indented with two tabulations using `"\t\t"`, or four spaces using `"    "`.

String **insert**(position: int, what: String) const

Inserts `what` at the given `position` in the string.

---

bool **is_absolute_path**() const

Returns `true` if the string is a path to a file or directory, and its starting point is explicitly defined. This method is the opposite of is_relative_path.

This includes all paths starting with `"res://"`, `"user://"`, `"C:\"`, `"/"`, etc.

bool **is_empty**() const

Returns `true` if the string's length is `0` (`""`). See also length.

bool **is_relative_path**() const

Returns `true` if the string is a path, and its starting point is dependent on context. The path could begin from the current directory, or the current Node (if the string is derived from a NodePath), and may sometimes be prefixed with `"./"`. This method is the opposite of is_absolute_path.

bool **is_subsequence_of**(text: String) const

Returns `true` if all characters of this string can be found in `text` in their original order.

```
var text = "Wow, incredible!"

print("inedible".is_subsequence_of(text)) # Prints true
print("Word!".is_subsequence_of(text))    # Prints true
print("Window".is_subsequence_of(text))   # Prints false
print("".is_subsequence_of(text))         # Prints true
```

bool **is_subsequence_ofn**(text: String) const

Returns `true` if all characters of this string can be found in `text` in their original order, **ignoring case**.

bool **is_valid_filename**() const 🔗

Returns `true` if this string does not contain characters that are not allowed in file names (`: / \ ? * " | % < >`).

---

bool **is_valid_float**() const 🔗

Returns `true` if this string represents a valid floating-point number. A valid float may contain only digits, one decimal point (`.`), and the exponent letter (`e`). It may also be prefixed with a positive (`+`) or negative (`-`) sign. Any valid integer is also a valid float (see is_valid_int). See also to_float.

```
print("1.7".is_valid_float())   # Prints true
print("24".is_valid_float())    # Prints true
print("7e3".is_valid_float())   # Prints true
print("Hello".is_valid_float()) # Prints false
```

---

bool **is_valid_hex_number**(with_prefix: bool = false) const 🔗

Returns `true` if this string is a valid hexadecimal number. A valid hexadecimal number only contains digits or letters `A` to `F` (either uppercase or lowercase), and may be prefixed with a positive (`+`) or negative (`-`) sign.

If `with_prefix` is `true`, the hexadecimal number needs to prefixed by `"0x"` to be considered valid.

```
print("A08E".is_valid_hex_number())     # Prints true
print("-AbCdEf".is_valid_hex_number()) # Prints true
print("2.5".is_valid_hex_number())      # Prints false

print("0xDEADC0DE".is_valid_hex_number(true)) # Prints true
```

---

bool **is_valid_html_color**() const 🔗

Returns `true` if this string is a valid color in hexadecimal HTML notation. The string must be a hexadecimal value (see is_valid_hex_number) of either 3, 4, 6 or 8 digits, and may be prefixed by a hash sign (`#`). Other HTML notations for colors, such as names or `hsl()`, are not considered valid. See also Color.html.

---

bool **is_valid_identifier**() const

Returns `true` if this string is a valid identifier. A valid identifier may contain only letters, digits and underscores (`_`), and the first character may not be a digit.

```
print("node_2d".is_valid_identifier())    # Prints true
print("TYPE_FLOAT".is_valid_identifier()) # Prints true
print("1st_method".is_valid_identifier()) # Prints false
print("MyMethod#2".is_valid_identifier()) # Prints false
```

---

bool **is_valid_int**() const

Returns `true` if this string represents a valid integer. A valid integer only contains digits, and may be prefixed with a positive (`+`) or negative (`-`) sign. See also to_int.

```
print("7".is_valid_int())    # Prints true
print("1.65".is_valid_int()) # Prints false
print("Hi".is_valid_int())   # Prints false
print("+3".is_valid_int())   # Prints true
print("-12".is_valid_int())  # Prints true
```

---

bool **is_valid_ip_address**() const

Returns `true` if this string represents a well-formatted IPv4 or IPv6 address. This method considers reserved IP addresses [https://en.wikipedia.org/wiki/Reserved_IP_addresses] such as `"0.0.0.0"` and `"ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff"` as valid.

String **join**(parts: PackedStringArray) const

Returns the concatenation of `parts`' elements, with each element separated by the string calling this method. This method is the opposite of split.

Example:

GDScript

```gdscript
var fruits = ["Apple", "Orange", "Pear", "Kiwi"]

print(", ".join(fruits))  # Prints "Apple, Orange, Pear, Kiwi"
print("---".join(fruits)) # Prints "Apple---Orange---Pear---Kiwi"
```

C#

```csharp
var fruits = new string[] {"Apple", "Orange", "Pear", "Kiwi"};

// In C#, this method is static.
GD.Print(string.Join(", ", fruits));  // Prints "Apple, Orange, Pear, Kiwi"
GD.Print(string.Join("---", fruits)); // Prints "Apple---Orange---Pear---Kiwi"
```

String **json_escape**() const

Returns a copy of the string with special characters escaped using the JSON standard. Because it closely matches the C standard, it is possible to use c_unescape to unescape the string, if necessary.

String **left**(length: int) const

Returns the first `length` characters from the beginning of the string. If `length` is negative, strips the last `length` characters from the string's end.

```
print("Hello World!".left(3))  # Prints "Hel"
print("Hello World!".left(-4)) # Prints "Hello Wo"
```

---

int **length**() const

Returns the number of characters in the string. Empty strings (`""`) always return `0`. See also is_empty.

String **lpad**(min_length: int, character: String = " ") const

Formats the string to be at least `min_length` long by adding `character`s to the left of the string, if necessary. See also rpad.

String **lstrip**(chars: String) const

Removes a set of characters defined in `chars` from the string's beginning. See also rstrip.

Note: `chars` is not a prefix. Use trim_prefix to remove a single prefix, rather than a set of characters.

---

bool **match**(expr: String) const

Does a simple expression match (also called "glob" or "globbing"), where `*` matches zero or more arbitrary characters and `?` matches any single character except a period (`.`). An empty string or empty expression always evaluates to `false`.

---

bool **matchn**(expr: String) const

Does a simple **case-insensitive** expression match, where `*` matches zero or more arbitrary characters and `?` matches any single character except a period (`.`). An empty string or empty expression always evaluates to `false`.

PackedByteArray **md5_buffer**() const 🔗

Returns the MD5 hash [https://en.wikipedia.org/wiki/MD5] of the string as a PackedByteArray.

String **md5_text**() const 🔗

Returns the MD5 hash [https://en.wikipedia.org/wiki/MD5] of the string as another **String**.

---

int **naturalcasecmp_to**(to: String) const 🔗

Performs a **case-sensitive**, *natural order* comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "Less than" or "greater than" are determined by the Unicode code points [https://en.wikipedia.org/wiki/List_of_Unicode_characters] of each string, which roughly matches the alphabetical order.

When used for sorting, natural order comparison orders sequences of numbers by the combined value of each digit as is often expected, instead of the single digit's value. A sorted sequence of numbered strings will be `["1", "2", "3", ...]`, not `["1", "10", "2", "3", ...]`.

With different string lengths, returns `1` if this string is longer than the `to` string, or `-1` if shorter. Note that the length of empty strings is *always* `0`.

To get a bool result from a string comparison, use the `==` operator instead. See also naturalnocasecmp_to, filecasecmp_to, and nocasecmp_to.

---

int **naturalnocasecmp_to**(to: String) const 🔗

Performs a **case-insensitive**, *natural order* comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "Less than" or "greater than" are determined by the Unicode code points [https://en.wikipedia.org/wiki/List_of_Unicode_characters] of each string, which roughly matches the alphabetical order. Internally, lowercase characters are converted to uppercase for the comparison.

When used for sorting, natural order comparison orders sequences of numbers by the combined value of each digit as is often expected, instead of the single digit's value. A sorted sequence of numbered strings will be `["1", "2", "3", ...]`, not `["1", "10", "2", "3", ...]`.

With different string lengths, returns `1` if this string is longer than the `to` string, or `-1` if shorter. Note that the length of empty strings is *always* `0`.

To get a bool result from a string comparison, use the `==` operator instead. See also naturalcasecmp_to, filenocasecmp_to, and casecmp_to.

---

int **nocasecmp_to**(to: String) const 🔗

Performs a **case-insensitive** comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "Less than" or "greater than" are determined by the Unicode code points [https://en.wikipedia.org/wiki/List_of_Unicode_characters] of each string, which roughly matches the alphabetical order. Internally, lowercase characters are converted to uppercase for the comparison.

With different string lengths, returns `1` if this string is longer than the `to` string, or `-1` if shorter. Note that the length of empty strings is *always* `0`.

To get a bool result from a string comparison, use the `==` operator instead. See also casecmp_to, filenocasecmp_to, and naturalnocasecmp_to.

String **num**(number: float, decimals: int = -1) static

Converts a float to a string representation of a decimal number, with the number of decimal places specified in `decimals`.

If `decimals` is `-1` as by default, the string representation may only have up to 14 significant digits, with digits before the decimal point having priority over digits after.

Trailing zeros are not included in the string. The last digit is rounded, not truncated.

Example:

```
String.num(3.141593)      # Returns "3.141593"
String.num(3.141593, 3)   # Returns "3.142"
String.num(3.14159300)    # Returns "3.141593"

# Here, the last digit will be rounded up,
# which reduces the total digit count, since trailing zeros are
removed:
String.num(42.129999, 5) # Returns "42.13"

# If `decimals` is not specified, the maximum number of
significant digits is 14:
String.num(-0.0000012345432123454321)     # Returns
"-0.00000123454321"
```

```
String.num(-10000.0000012345432123454321) # Returns
"-10000.0000012345"
```

String **num_int64**(number: int, base: int = 10, capitalize_hex: bool = false) static

Converts the given `number` to a string representation, with the given `base`.

By default, `base` is set to decimal (`10`). Other common bases in programming include binary (`2`), octal [https://en.wikipedia.org/wiki/Octal] (`8`), hexadecimal (`16`).

If `capitalize_hex` is `true`, digits higher than 9 are represented in uppercase.

String **num_scientific**(number: float) static

Converts the given `number` to a string representation, in scientific notation.

GDScript

```
var n = -5.2e8
print(n)                          # Prints -520000000
print(String.num_scientific(n))  # Prints -5.2e+08
```

C#

```
// This method is not implemented in C#.
// Use `string.ToString()` with "e" to achieve similar results.
var n = -5.2e8f;
GD.Print(n);                   // Prints -520000000
GD.Print(n.ToString("e1")); // Prints -5.2e+008
```

Note: In C#, this method is not implemented. To achieve similar results, see C#'s Standard numeric format strings [https://learn.microsoft.com/en-us/dotnet/standard/base-types/standard-numeric-format-strings]

String **num_uint64**(number: int, base: int = 10, capitalize_hex: bool = false) static 🔗

Converts the given unsigned int to a string representation, with the given `base`.

By default, `base` is set to decimal (`10`). Other common bases in programming include binary (`2`), octal [https://en.wikipedia.org/wiki/Octal] (`8`), hexadecimal (`16`).

If `capitalize_hex` is `true`, digits higher than 9 are represented in uppercase.

String **pad_decimals**(digits: int) const 🔗

Formats the string representing a number to have an exact number of `digits` *after* the decimal point.

String **pad_zeros**(digits: int) const 🔗

Formats the string representing a number to have an exact number of `digits` *before* the decimal point.

String **path_join**(file: String) const 🔗

Concatenates `file` at the end of the string as a subpath, adding `/` if necessary.

Example: `"this/is".path_join("path") == "this/is/path"`.

String **repeat**(count: int) const 🔗

Repeats this string a number of times. `count` needs to be greater than `0`. Otherwise, returns an empty string.

String **replace**(what: String, forwhat: String) const 🔗

Replaces all occurrences of `what` inside the string with the given `forwhat`.

String **replacen**(what: String, forwhat: String) const 🔗

Replaces all **case-insensitive** occurrences of `what` inside the string with the given `forwhat`.

String **reverse**() const 🔗

Returns the copy of this string in reverse order. This operation works on unicode codepoints, rather than sequences of codepoints, and may break things like compound letters or emojis.

---

int **rfind**(what: String, from: int = -1) const 🔗

Returns the index of the **last** occurrence of `what` in this string, or `-1` if there are none. The search's start can be specified with `from`, continuing to the beginning of the string. This method is the reverse of find.

---

int **rfindn**(what: String, from: int = -1) const

Returns the index of the **last case-insensitive** occurrence of `what` in this string, or `-1` if there are none. The starting search index can be specified with `from`, continuing to the beginning of the string. This method is the reverse of findn.

String **right**(length: int) const

Returns the last `length` characters from the end of the string. If `length` is negative, strips the first `length` characters from the string's beginning.

```
print("Hello World!".right(3))  # Prints "ld!"
print("Hello World!".right(-4)) # Prints "o World!"
```

String **rpad**(min_length: int, character: String = " ") const

Formats the string to be at least `min_length` long, by adding `character`s to the right of the string, if necessary. See also lpad.

PackedStringArray **rsplit**(delimiter: String = "", allow_empty: bool = true, maxsplit: int = 0) const

Splits the string using a `delimiter` and returns an array of the substrings, starting from the end of the string. The splits in the returned array appear in the same order as the original string. If `delimiter` is an empty string, each substring will be a single character.

If `allow_empty` is `false`, empty strings between adjacent delimiters are excluded from the array.

If `maxsplit` is greater than `0`, the number of splits may not exceed `maxsplit`. By default, the entire string is split, which is mostly identical to split.

Example:

GDScript

```
var some_string = "One,Two,Three,Four"
var some_array = some_string.rsplit(",", true, 1)

print(some_array.size()) # Prints 2
print(some_array[0])     # Prints "One,Two,Three"
print(some_array[1])     # Prints "Four"
```

C#

```
// In C#, there is no String.RSplit() method.
```

String **rstrip**(chars: String) const 🔗

Removes a set of characters defined in `chars` from the string's end. See also lstrip.

Note: `chars` is not a suffix. Use trim_suffix to remove a single suffix, rather than a set of characters.

PackedByteArray **sha1_buffer**() const 🔗

Returns the SHA-1 [https://en.wikipedia.org/wiki/SHA-1] hash of the string as a PackedByteArray.

String **sha1_text**() const 🔗

Returns the SHA-1 [https://en.wikipedia.org/wiki/SHA-1] hash of the string as another **String**.

PackedByteArray **sha256_buffer**() const

Returns the SHA-256 [https://en.wikipedia.org/wiki/SHA-2] hash of the string as a PackedByteArray.

String **sha256_text**() const

Returns the SHA-256 [https://en.wikipedia.org/wiki/SHA-2] hash of the string as another **String**.

---

float **similarity**(text: String) const

Returns the similarity index (Sorensen-Dice coefficient [https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient]) of this string compared to another. A result of `1.0` means totally similar, while `0.0` means totally dissimilar.

```
print("ABC123".similarity("ABC123")) # Prints 1.0
print("ABC123".similarity("XYZ456")) # Prints 0.0
print("ABC123".similarity("123ABC")) # Prints 0.8
print("ABC123".similarity("abc123")) # Prints 0.4
```

String **simplify_path**() const

If the string is a valid file path, converts the string into a canonical path. This is the shortest possible path, without `"./"`, and all the unnecessary `".."` and `"/"`.

```
var simple_path = "./path/to///../file".simplify_path()
print(simple_path) # Prints "path/file"
```

PackedStringArray **split**(delimiter: String = "", allow_empty: bool = true, maxsplit: int = 0) const

Splits the string using a `delimiter` and returns an array of the substrings. If `delimiter` is an empty string, each substring will be a single character. This method is the opposite of join.

If `allow_empty` is `false`, empty strings between adjacent delimiters are excluded from the array.

If `maxsplit` is greater than `0`, the number of splits may not exceed `maxsplit`. By default, the entire string is split.

Example:

GDScript

```
var some_array = "One,Two,Three,Four".split(",", true, 2)

print(some_array.size()) # Prints 3
print(some_array[0])     # Prints "One"
print(some_array[1])     # Prints "Two"
print(some_array[2])     # Prints "Three,Four"
```

C#

```
// C#'s `Split()` does not support the `maxsplit` parameter.
var someArray = "One,Two,Three".Split(",");

GD.Print(someArray[0]); // Prints "One"
GD.Print(someArray[1]); // Prints "Two"
GD.Print(someArray[2]); // Prints "Three"
```

Note: If you only need one substring from the array, consider using get_slice which is faster. If you need to split strings with more complex rules, use the RegEx class instead.

PackedFloat64Array **split_floats**(delimiter: String, allow_empty: bool = true) const

Splits the string into floats by using a `delimiter` and returns a PackedFloat64Array.

If `allow_empty` is `false`, empty or invalid float conversions between adjacent delimiters are excluded.

```
var a = "1,2,4.5".split_floats(",")         # a is [1.0, 2.0, 4.5]
var c = "1| ||4.5".split_floats("|")        # c is [1.0, 0.0, 0.0, 4.5]
var b = "1| ||4.5".split_floats("|", false) # b is [1.0, 4.5]
```

String **strip_edges**(left: bool = true, right: bool = true) const

Strips all non-printable characters from the beginning and the end of the string. These include spaces, tabulations (`\t`), and newlines (`\n` `\r`).

If `left` is `false`, ignores the string's beginning. Likewise, if `right` is `false`, ignores the string's end.

String **strip_escapes**() const

Strips all escape characters from the string. These include all non-printable control characters of the first page of the ASCII table (values from 0 to 31), such as tabulation (`\t`) and newline (`\n`, `\r`) characters, but *not* spaces.

String **substr**(from: int, len: int = -1) const

Returns part of the string from the position `from` with length `len`. If `len` is `-1` (as by default), returns the rest of the string starting from the given position.

PackedByteArray **to_ascii_buffer**() const

Converts the string to an ASCII [https://en.wikipedia.org/wiki/ASCII]/Latin-1 encoded PackedByteArray. This method is slightly faster than to_utf8_buffer, but replaces all unsupported characters with spaces. This is the inverse of PackedByteArray.get_string_from_ascii.

String **to_camel_case**() const

Returns the string converted to `camelCase`.

---

float **to_float**() const

Converts the string representing a decimal number into a float. This method stops on the first non-number character, except the first decimal point (`.`) and the exponent letter (`e`). See also is_valid_float.

```
var a = "12.35".to_float()  # a is 12.35
var b = "1.2.3".to_float()  # b is 1.2
var c = "12xy3".to_float()  # c is 12.0
var d = "1e3".to_float()    # d is 1000.0
var e = "Hello!".to_float() # e is 0.0
```

---

int **to_int**() const

Converts the string representing an integer number into an int. This method removes any non-number character and stops at the first decimal point (`.`). See also is_valid_int.

```
var a = "123".to_int()     # a is 123
var b = "x1y2z3".to_int() # b is 123
var c = "-1.2.3".to_int() # c is -1
var d = "Hello!".to_int() # d is 0
```

String **to_lower**() const

Returns the string converted to `lowercase`.

String **to_pascal_case**() const

Returns the string converted to `PascalCase`.

String **to_snake_case**() const

Returns the string converted to `snake_case`.

Note: Numbers followed by a *single* letter are not separated in the conversion to keep some words (such as "2D") together.

GDScript

```
"Node2D".to_snake_case()               # Returns "node_2d"
"2nd place".to_snake_case()            # Returns "2_nd_place"
"Texture3DAssetFolder".to_snake_case() # Returns
"texture_3d_asset_folder"
```

C#

```
"Node2D".ToSnakeCase();               // Returns "node_2d"
"2nd place".ToSnakeCase();            // Returns "2_nd_place"
"Texture3DAssetFolder".ToSnakeCase(); // Returns
"texture_3d_asset_folder"
```

String **to_upper**() const

Returns the string converted to `UPPERCASE`.

PackedByteArray **to_utf8_buffer**() const 🔗

Converts the string to a UTF-8 [https://en.wikipedia.org/wiki/UTF-8] encoded PackedByteArray. This method is slightly slower than to_ascii_buffer, but supports all UTF-8 characters. For most cases, prefer using this method. This is the inverse of PackedByteArray.get_string_from_utf8.

PackedByteArray **to_utf16_buffer**() const 🔗

Converts the string to a UTF-16 [https://en.wikipedia.org/wiki/UTF-16] encoded PackedByteArray. This is the inverse of PackedByteArray.get_string_from_utf16.

PackedByteArray **to_utf32_buffer**() const 🔗

Converts the string to a UTF-32 [https://en.wikipedia.org/wiki/UTF-32] encoded PackedByteArray. This is the inverse of PackedByteArray.get_string_from_utf32.

PackedByteArray **to_wchar_buffer**() const 🔗

Converts the string to a wide character [https://en.wikipedia.org/wiki/Wide_character] (`wchar_t`, UTF-16 on Windows, UTF-32 on other platforms) encoded PackedByteArray. This is the inverse of PackedByteArray.get_string_from_wchar.

String **trim_prefix**(prefix: String) const 🔗

Removes the given `prefix` from the start of the string, or returns the string unchanged.

String **trim_suffix**(suffix: String) const

Removes the given `suffix` from the end of the string, or returns the string unchanged.

---

int **unicode_at**(at: int) const

Returns the character code at position `at`.

String **uri_decode**() const

Decodes the string from its URL-encoded format. This method is meant to properly decode the parameters in a URL when receiving an HTTP request. See also uri_encode.

GDScript

```
var url = "$DOCS_URL/?highlight=Godot%20Engine%3%docs"
print(url.uri_decode()) # Prints "$DOCS_URL/?highlight=Godot Engine:docs"
```

C#

```
var url = "$DOCS_URL/?highlight=Godot%20Engine%3%docs"
GD.Print(url.URIDecode()) // Prints "$DOCS_URL/?highlight=Godot Engine:docs"
```

String **uri_encode**() const

Encodes the string to URL-friendly format. This method is meant to properly encode the parameters in a URL when sending an HTTP request. See also uri_decode.

GDScript

```
var prefix = "$DOCS_URL/?highlight="
var url = prefix + "Godot Engine:docs".uri_encode()

print(url) # Prints "$DOCS_URL/?highlight=Godot%20Engine%3%docs"
```

C#

```
var prefix = "$DOCS_URL/?highlight=";
var url = prefix + "Godot Engine:docs".URIEncode();

GD.Print(url); // Prints "$DOCS_URL/?highlight=Godot%20Engine%3%docs"
```

String **validate_filename**() const

Returns a copy of the string with all characters that are not allowed in is_valid_filename replaced with underscores.

String **validate_node_name**() const

Returns a copy of the string with all characters that are not allowed in Node.name (`. : @ / " %`) replaced with underscores.

String **xml_escape**(escape_quotes: bool = false) const

Returns a copy of the string with special characters escaped using the XML standard. If `escape_quotes` is `true`, the single quote (`'`) and double quote (`"`) characters are also escaped.

String **xml_unescape**() const

Returns a copy of the string with escaped characters replaced by their meanings according to the XML standard.

Operator Descriptions

bool **operator !=**(right: String)

Returns `true` if both strings do not contain the same sequence of characters.

---

bool **operator !=**(right: StringName)

Returns `true` if this **String** is not equivalent to the given StringName.

String **operator %**(right: Variant)

Formats the **String**, replacing the placeholders with one or more parameters. To pass multiple parameters, `right` needs to be an Array.

```
print("I caught %d fishes!" % 2) # Prints "I caught 2 fishes!"

var my_message = "Travelling to %s, at %2.2f km/h."
var location = "Deep Valley"
var speed = 40.3485
print(my_message % [location, speed]) # Prints "Travelling to
Deep Valley, at 40.35 km/h."
```

For more information, see the GDScript format strings tutorial.

Note: In C#, this operator is not available. Instead, see how to interpolate strings with "$" [https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated].

String **operator +**(right: String)

Appends `right` at the end of this **String**, also known as a string concatenation.

String **operator +**(right: StringName)

Appends `right` at the end of this **String**, returning a **String**. This is also known as a string concatenation.

bool **operator <**(right: String)

Returns `true` if the left **String** comes before `right` in Unicode order [https://en.wikipedia.org/wiki/List_of_Unicode_characters], which roughly matches the alphabetical order. Useful for sorting.

bool **operator <=**(right: String)

Returns `true` if the left **String** comes before `right` in Unicode order [https://en.wikipedia.org/wiki/List_of_Unicode_characters], which roughly matches the alphabetical order, or if both are equal.

bool **operator ==**(right: String)

Returns `true` if both strings contain the same sequence of characters.

bool **operator ==**(right: StringName) 🔗

Returns `true` if this **String** is equivalent to the given StringName.

---

bool **operator >**(right: String) 🔗

Returns `true` if the left **String** comes after `right` in Unicode order [https://en.wikipedia.org/wiki/List_of_Unicode_characters], which roughly matches the alphabetical order. Useful for sorting.

---

bool **operator >=**(right: String) 🔗

Returns `true` if the left **String** comes after `right` in Unicode order [https://en.wikipedia.org/wiki/List_of_Unicode_characters], which roughly matches the alphabetical order, or if both are equal.

String **operator []**(index: int) 🔗

Returns a new **String** that only contains the character at `index`. Indices start from `0`. If `index` is greater or equal to `0`, the character is fetched starting from the beginning of the string. If `index` is a negative value, it is fetched starting from the end. Accessing a string out-of-bounds will cause a run-time error, pausing the project execution if run from the editor.

StringName

A built-in type for unique strings.

Description

StringNames are immutable strings designed for general-purpose representation of unique names (also called "string interning"). Two **StringName**s with the same value are the same object. Comparing them is extremely fast compared to regular Strings.

You will usually pass a String to methods expecting a **StringName** and it will be automatically converted (often at compile time), but in rare cases you can construct a **StringName** ahead of time with the **StringName** constructor or, in GDScript, the literal syntax `&"example"`. Manually constructing a **StringName** allows you to control when the conversion from String occurs or to use the literal and prevent conversions entirely.

See also NodePath, which is a similar concept specifically designed to store pre-parsed scene tree paths.

All of String's methods are available in this class too. They convert the **StringName** into a string, and they also return a string. This is highly inefficient and should only be used if the string is desired.

Note: In C#, an explicit conversion to `System.String` is required to use the methods listed on this page. Use the `ToString()` method to cast a **StringName** to a string, and then use the equivalent methods in `System.String` or `StringExtensions`.

Note: In a boolean context, a **StringName** will evaluate to `false` if it is empty (`StringName("")`). Otherwise, a **StringName** will always evaluate to `true`. The `not` operator cannot be used. Instead, is_empty should be used to check for empty **StringName**s.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Constructors

StringName	StringName()
StringName	StringName(from: StringName)
StringName	StringName(from: String)

Methods

bool	begins_with(text: String) const
PackedStringArray	bigrams() const
int	bin_to_int() const
String	c_escape() const

String	c_unescape() const
String	capitalize() const
int	casecmp_to(to: String) const
bool	contains(what: String) const
bool	containsn(what: String) const
int	count(what: String, from: int = 0, to: int = 0) const
int	countn(what: String, from: int = 0, to: int = 0) const
String	dedent() const
bool	ends_with(text: String) const
String	erase(position: int, chars: int = 1) const
int	filecasecmp_to(to: String) const
int	filenocasecmp_to(to: String) const

int	find(what: String, from: int = 0) const
int	findn(what: String, from: int = 0) const
String	format(values: Variant, placeholder: String = "{_}") const
String	get_base_dir() const
String	get_basename() const
String	get_extension() const
String	get_file() const
String	get_slice(delimiter: String, slice: int) const
int	get_slice_count(delimiter: String) const
String	get_slicec(delimiter: int, slice: int) const
int	hash() const
PackedByteArray	hex_decode() const

int	hex_to_int() const
String	indent(prefix: String) const
String	insert(position: int, what: String) const
bool	is_absolute_path() const
bool	is_empty() const
bool	is_relative_path() const
bool	is_subsequence_of(text: String) const
bool	is_subsequence_ofn(text: String) const
bool	is_valid_filename() const
bool	is_valid_float() const
bool	is_valid_hex_number(with_prefix: bool = false) const
bool	is_valid_html_color() const

bool	is_valid_identifier() const
bool	is_valid_int() const
bool	is_valid_ip_address() const
String	join(parts: PackedStringArray) const
String	json_escape() const
String	left(length: int) const
int	length() const
String	lpad(min_length: int, character: String = " ") const
String	lstrip(chars: String) const
bool	match(expr: String) const
bool	matchn(expr: String) const
PackedByteArray	md5_buffer() const

String	md5_text() const
int	naturalcasecmp_to(to: String) const
int	naturalnocasecmp_to(to: String) const
int	nocasecmp_to(to: String) const
String	pad_decimals(digits: int) const
String	pad_zeros(digits: int) const
String	path_join(file: String) const
String	repeat(count: int) const
String	replace(what: String, forwhat: String) const
String	replacen(what: String, forwhat: String) const
String	reverse() const
int	rfind(what: String, from: int = -1) const
int	rfindn(what: String, from: int = -1) const

String	right(length: int) const
String	rpad(min_length: int, character: String = " ") const
PackedStringArray	rsplit(delimiter: String = "", allow_empty: bool = true, maxsplit: int = 0) const
String	rstrip(chars: String) const
PackedByteArray	sha1_buffer() const
String	sha1_text() const
PackedByteArray	sha256_buffer() const
String	sha256_text() const
float	similarity(text: String) const
String	simplify_path() const
PackedStringArray	split(delimiter: String = "", allow_empty: bool = true, maxsplit: int = 0) const

PackedFloat64Array	split_floats(delimiter: String, allow_empty: bool = true) const
String	strip_edges(left: bool = true, right: bool = true) const
String	strip_escapes() const
String	substr(from: int, len: int = -1) const
PackedByteArray	to_ascii_buffer() const
String	to_camel_case() const
float	to_float() const
int	to_int() const
String	to_lower() const
String	to_pascal_case() const
String	to_snake_case() const
String	to_upper() const

PackedByteArray	to_utf8_buffer() const
PackedByteArray	to_utf16_buffer() const
PackedByteArray	to_utf32_buffer() const
PackedByteArray	to_wchar_buffer() const
String	trim_prefix(prefix: String) const
String	trim_suffix(suffix: String) const
int	unicode_at(at: int) const
String	uri_decode() const
String	uri_encode() const
String	validate_filename() const
String	validate_node_name() const
String	xml_escape(escape_quotes: bool = false) const

String	xml_unescape() const

Operators

bool	operator !=(right: String)
bool	operator !=(right: StringName)
String	operator %(right: Variant)
String	operator +(right: String)
String	operator +(right: StringName)
bool	operator <(right: StringName)
bool	operator <=(right: StringName)
bool	operator ==(right: String)
bool	operator ==(right: StringName)
bool	operator >(right: StringName)

bool	operator >=(right: StringName)

Constructor Descriptions

StringName **StringName**() 🔗

Constructs an empty **StringName**.

StringName **StringName**(from: StringName)

Constructs a **StringName** as a copy of the given **StringName**.

StringName **StringName**(from: String)

Creates a new **StringName** from the given String. In GDScript, `StringName("example")` is equivalent to `&"example"`.

Method Descriptions

bool **begins_with**(text: String) const 🔗

Returns `true` if the string begins with the given `text`. See also ends_with.

PackedStringArray **bigrams**() const 🔗

Returns an array containing the bigrams (pairs of consecutive characters) of this string.

```
print("Get up!".bigrams()) # Prints ["Ge", "et", "t ", " u",
"up", "p!"]
```

---

int **bin_to_int**() const

Converts the string representing a binary number into an int. The string may optionally be prefixed with `"0b"`, and an additional `-` prefix for negative numbers.

GDScript

```
print("101".bin_to_int())   # Prints 5
print("0b101".bin_to_int()) # Prints 5
print("-0b10".bin_to_int()) # Prints -2
```

C#

```
GD.Print("101".BinToInt());   // Prints 5
GD.Print("0b101".BinToInt()); // Prints 5
GD.Print("-0b10".BinToInt()); // Prints -2
```

String **c_escape**() const

Returns a copy of the string with special characters escaped using the C language standard.

String **c_unescape**() const

Returns a copy of the string with escaped characters replaced by their meanings. Supported escape sequences are `\'`, `\"`, `\\`, `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v`.

Note: Unlike the GDScript parser, this method doesn't support the `\uXXXX` escape sequence.

String **capitalize**() const

Changes the appearance of the string: replaces underscores (_) with spaces, adds spaces before uppercase letters in the middle of a word, converts all letters to lowercase, then converts the first one and each one following a space to uppercase.

GDScript

```
"move_local_x".capitalize()    # Returns "Move Local X"
"sceneFile_path".capitalize() # Returns "Scene File Path"
"2D, FPS, PNG".capitalize()    # Returns "2d, Fps, Png"
```

C#

```
"move_local_x".Capitalize();    // Returns "Move Local X"
"sceneFile_path".Capitalize(); // Returns "Scene File Path"
"2D, FPS, PNG".Capitalize();    // Returns "2d, Fps, Png"
```

---

int **casecmp_to**(to: String) const

Performs a case-sensitive comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "Less than" and "greater than" are determined by the Unicode code points [https://en.wikipedia.org/wiki/List_of_Unicode_characters] of each string, which roughly matches the alphabetical order.

With different string lengths, returns `1` if this string is longer than the `to` string, or `-1` if shorter. Note that the length of empty strings is *always* `0`.

To get a bool result from a string comparison, use the `==` operator instead. See also nocasecmp_to, filecasecmp_to, and naturalcasecmp_to.

---

bool **contains**(what: String) const

Returns `true` if the string contains `what`. In GDScript, this corresponds to the `in` operator.

GDScript

```
print("Node".contains("de")) # Prints true
print("team".contains("I"))  # Prints false
print("I" in "team")         # Prints false
```

C#

```
GD.Print("Node".Contains("de")); // Prints true
GD.Print("team".Contains("I"));  // Prints false
```

If you need to know where `what` is within the string, use find. See also containsn.

---

bool **containsn**(what: String) const 🔗

Returns `true` if the string contains `what`, **ignoring case**.

If you need to know where `what` is within the string, use findn. See also contains.

---

int **count**(what: String, from: int = 0, to: int = 0) const 🔗

Returns the number of occurrences of the substring `what` between `from` and `to` positions. If `to` is 0, the search continues until the end of the string.

---

int **countn**(what: String, from: int = 0, to: int = 0) const 🔗

Returns the number of occurrences of the substring `what` between `from` and `to` positions, **ignoring case**. If `to` is 0, the search continues until the end of the string.

String **dedent**() const

Returns a copy of the string with indentation (leading tabs and spaces) removed. See also indent to add indentation.

bool **ends_with**(text: String) const

Returns `true` if the string ends with the given `text`. See also begins_with.

String **erase**(position: int, chars: int = 1) const

Returns a string with `chars` characters erased starting from `position`. If `chars` goes beyond the string's length given the specified `position`, fewer characters will be erased from the returned string. Returns an empty string if either `position` or `chars` is negative. Returns the original string unmodified if `chars` is `0`.

int **filecasecmp_to**(to: String) const

Like naturalcasecmp_to but prioritizes strings that begin with periods (`.`) and underscores (`_`) before any other character. Useful when sorting folders or file names.

To get a bool result from a string comparison, use the `==` operator instead. See also filenocasecmp_to, naturalcasecmp_to, and casecmp_to.

int **filenocasecmp_to**(to: String) const

Like naturalnocasecmp_to but prioritizes strings that begin with periods (`.`) and underscores (`_`) before any other character. Useful when sorting folders or file names.

To get a bool result from a string comparison, use the `==` operator instead. See also filecasecmp_to, naturalnocasecmp_to, and nocasecmp_to.

---

int **find**(what: String, from: int = 0) const

Returns the index of the **first** occurrence of `what` in this string, or `-1` if there are none. The search's start can be specified with `from`, continuing to the end of the string.

GDScript

```
print("Team".find("I")) # Prints -1

print("Potato".find("t"))    # Prints 2
print("Potato".find("t", 3)) # Prints 4
print("Potato".find("t", 5)) # Prints -1
```

C#

```
GD.Print("Team".Find("I")); // Prints -1

GD.Print("Potato".Find("t"));    // Prints 2
GD.Print("Potato".Find("t", 3)); // Prints 4
GD.Print("Potato".Find("t", 5)); // Prints -1
```

Note: If you just want to know whether the string contains `what`, use contains. In GDScript, you may also use the `in` operator.

---

int **findn**(what: String, from: int = 0) const

Returns the index of the **first case-insensitive** occurrence of `what` in this string, or `-1` if there are none. The starting search index can be specified with `from`, continuing to the end of the string.

String **format**(values: Variant, placeholder: String = "{_}") const 🔗

Formats the string by replacing all occurrences of `placeholder` with the elements of `values`.

`values` can be a Dictionary or an Array. Any underscores in `placeholder` will be replaced with the corresponding keys in advance. Array elements use their index as keys.

```
# Prints "Waiting for Godot is a play by Samuel Beckett, and Godot Engine is named after it."
var use_array_values = "Waiting for {0} is a play by {1}, and {0} Engine is named after it."
print(use_array_values.format(["Godot", "Samuel Beckett"]))

# Prints "User 42 is Godot."
print("User {id} is {name}.".format({"id": 42, "name": "Godot"}))
```

Some additional handling is performed when `values` is an Array. If `placeholder` does not contain an underscore, the elements of the `values` array will be used to replace one occurrence of the placeholder in order; If an element of `values` is another 2-element array, it'll be interpreted as a key-value pair.

```
# Prints "User 42 is Godot."
print("User {} is {}.".format([42, "Godot"], "{}"))
print("User {id} is {name}.".format([["id", 42], ["name", "Godot"]]))
```

See also the GDScript format string tutorial.

Note: In C#, it's recommended to interpolate strings with "$" [https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated], instead.

String **get_base_dir**() const 🔗

If the string is a valid file path, returns the base directory name.

```
var dir_path = "/path/to/file.txt".get_base_dir() # dir_path is
"/path/to"
```

String **get_basename**() const 🔗

If the string is a valid file path, returns the full file path, without the extension.

```
var base = "/path/to/file.txt".get_basename() # base is
"/path/to/file"
```

String **get_extension**() const 🔗

If the string is a valid file name or path, returns the file extension without the leading period (`.`). Otherwise, returns an empty string.

```
var a = "/path/to/file.txt".get_extension() # a is "txt"
var b = "cool.txt".get_extension()          # b is "txt"
var c = "cool.font.tres".get_extension()    # c is "tres"
var d = ".pack1".get_extension()            # d is "pack1"

var e = "file.txt.".get_extension()  # e is ""
var f = "file.txt..".get_extension() # f is ""
var g = "txt".get_extension()        # g is ""
var h = "".get_extension()           # h is ""
```

String **get_file**() const 🔗

If the string is a valid file path, returns the file name, including the extension.

```
var file = "/path/to/icon.png".get_file() # file is "icon.png"
```

String **get_slice**(delimiter: String, slice: int) const

Splits the string using a `delimiter` and returns the substring at index `slice`. Returns an empty string if the `slice` does not exist.

This is faster than split, if you only need one substring.

Example:

```
print("i/am/example/hi".get_slice("/", 2)) # Prints "example"
```

---

int **get_slice_count**(delimiter: String) const

Returns the total number of slices when the string is split with the given `delimiter` (see split).

String **get_slicec**(delimiter: int, slice: int) const

Splits the string using a Unicode character with code `delimiter` and returns the substring at index `slice`. Returns an empty string if the `slice` does not exist.

This is faster than split, if you only need one substring.

---

int **hash**() const

Returns the 32-bit hash value representing the string's contents.

Note: Strings with equal hash values are *not* guaranteed to be the same, as a result of hash collisions. On the contrary, strings with different hash values are guaranteed to be different.

PackedByteArray **hex_decode**() const

Decodes a hexadecimal string as a PackedByteArray.

GDScript

```
var text = "hello world"
var encoded = text.to_utf8_buffer().hex_encode() # outputs
"68656c6c6f20776f726c64"
print(buf.hex_decode().get_string_from_utf8())
```

C#

```
var text = "hello world";
var encoded = text.ToUtf8Buffer().HexEncode(); // outputs
"68656c6c6f20776f726c64"
GD.Print(buf.HexDecode().GetStringFromUtf8());
```

---

int **hex_to_int**() const

Converts the string representing a hexadecimal number into an int. The string may be optionally prefixed with `"0x"`, and an additional `-` prefix for negative numbers.

GDScript

```
print("0xff".hex_to_int()) # Prints 255
print("ab".hex_to_int())   # Prints 171
```

C#

```
GD.Print("0xff".HexToInt()); // Prints 255
GD.Print("ab".HexToInt());   // Prints 171
```

String **indent**(prefix: String) const

Indents every line of the string with the given `prefix`. Empty lines are not indented. See also dedent to remove indentation.

For example, the string can be indented with two tabulations using `"\t\t"`, or four spaces using `"    "`.

String **insert**(position: int, what: String) const

Inserts `what` at the given `position` in the string.

---

bool **is_absolute_path**() const

Returns `true` if the string is a path to a file or directory, and its starting point is explicitly defined. This method is the opposite of is_relative_path.

This includes all paths starting with `"res://"`, `"user://"`, `"C:\"`, `"/"`, etc.

---

bool **is_empty**() const

Returns `true` if the string's length is `0` (`""`). See also length.

---

bool **is_relative_path**() const

Returns `true` if the string is a path, and its starting point is dependent on context. The path could begin from the current directory, or the current Node (if the string is derived from a

NodePath), and may sometimes be prefixed with `"./"`. This method is the opposite of is_absolute_path.

---

bool **is_subsequence_of**(text: String) const 🔗

Returns `true` if all characters of this string can be found in `text` in their original order.

```
var text = "Wow, incredible!"

print("inedible".is_subsequence_of(text)) # Prints true
print("Word!".is_subsequence_of(text))    # Prints true
print("Window".is_subsequence_of(text))   # Prints false
print("".is_subsequence_of(text))         # Prints true
```

---

bool **is_subsequence_ofn**(text: String) const 🔗

Returns `true` if all characters of this string can be found in `text` in their original order, **ignoring case**.

---

bool **is_valid_filename**() const 🔗

Returns `true` if this string does not contain characters that are not allowed in file names (`:` `/` `\` `?` `*` `"` `|` `%` `<` `>`).

---

bool **is_valid_float**() const 🔗

Returns `true` if this string represents a valid floating-point number. A valid float may contain only digits, one decimal point (`.`), and the exponent letter (`e`). It may also be prefixed with a positive (`+`) or negative (`-`) sign. Any valid integer is also a valid float (see is_valid_int). See also to_float.

```
print("1.7".is_valid_float())   # Prints true
print("24".is_valid_float())    # Prints true
print("7e3".is_valid_float())   # Prints true
print("Hello".is_valid_float()) # Prints false
```

---

bool **is_valid_hex_number**(with_prefix: bool = false) const 🔗

Returns `true` if this string is a valid hexadecimal number. A valid hexadecimal number only contains digits or letters `A` to `F` (either uppercase or lowercase), and may be prefixed with a positive (`+`) or negative (`-`) sign.

If `with_prefix` is `true`, the hexadecimal number needs to prefixed by `"0x"` to be considered valid.

```
print("A08E".is_valid_hex_number())     # Prints true
print("-AbCdEf".is_valid_hex_number()) # Prints true
print("2.5".is_valid_hex_number())      # Prints false

print("0xDEADC0DE".is_valid_hex_number(true)) # Prints true
```

---

bool **is_valid_html_color**() const 🔗

Returns `true` if this string is a valid color in hexadecimal HTML notation. The string must be a hexadecimal value (see is_valid_hex_number) of either 3, 4, 6 or 8 digits, and may be prefixed by a hash sign (`#`). Other HTML notations for colors, such as names or `hsl()`, are not considered valid. See also Color.html.

---

bool **is_valid_identifier**() const 🔗

Returns `true` if this string is a valid identifier. A valid identifier may contain only letters, digits and underscores (`_`), and the first character may not be a digit.

```
print("node_2d".is_valid_identifier())    # Prints true
print("TYPE_FLOAT".is_valid_identifier()) # Prints true
print("1st_method".is_valid_identifier()) # Prints false
print("MyMethod#2".is_valid_identifier()) # Prints false
```

---

bool **is_valid_int**() const

Returns `true` if this string represents a valid integer. A valid integer only contains digits, and may be prefixed with a positive (`+`) or negative (`-`) sign. See also to_int.

```
print("7".is_valid_int())    # Prints true
print("1.65".is_valid_int()) # Prints false
print("Hi".is_valid_int())   # Prints false
print("+3".is_valid_int())   # Prints true
print("-12".is_valid_int())  # Prints true
```

---

bool **is_valid_ip_address**() const

Returns `true` if this string represents a well-formatted IPv4 or IPv6 address. This method considers reserved IP addresses [https://en.wikipedia.org/wiki/Reserved_IP_addresses] such as `"0.0.0.0"` and `"ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff"` as valid.

String **join**(parts: PackedStringArray) const

Returns the concatenation of `parts`' elements, with each element separated by the string calling this method. This method is the opposite of split.

Example:

GDScript

```
var fruits = ["Apple", "Orange", "Pear", "Kiwi"]
```

```
print(", ".join(fruits))   # Prints "Apple, Orange, Pear, Kiwi"
print("---".join(fruits)) # Prints "Apple---Orange---Pear---Kiwi"
```

C#

```
var fruits = new string[] {"Apple", "Orange", "Pear", "Kiwi"};

// In C#, this method is static.
GD.Print(string.Join(", ", fruits));  // Prints "Apple, Orange, Pear, Kiwi"
GD.Print(string.Join("---", fruits)); // Prints "Apple---Orange---Pear---Kiwi"
```

String **json_escape**() const

Returns a copy of the string with special characters escaped using the JSON standard. Because it closely matches the C standard, it is possible to use c_unescape to unescape the string, if necessary.

String **left**(length: int) const

Returns the first `length` characters from the beginning of the string. If `length` is negative, strips the last `length` characters from the string's end.

```
print("Hello World!".left(3))  # Prints "Hel"
print("Hello World!".left(-4)) # Prints "Hello Wo"
```

---

int **length**() const

Returns the number of characters in the string. Empty strings (`""`) always return `0`. See also is_empty.

String **lpad**(min_length: int, character: String = " ") const 🔗

Formats the string to be at least `min_length` long by adding `character`s to the left of the string, if necessary. See also rpad.

String **lstrip**(chars: String) const 🔗

Removes a set of characters defined in `chars` from the string's beginning. See also rstrip.

Note: `chars` is not a prefix. Use trim_prefix to remove a single prefix, rather than a set of characters.

---

bool **match**(expr: String) const 🔗

Does a simple expression match (also called "glob" or "globbing"), where `*` matches zero or more arbitrary characters and `?` matches any single character except a period (`.`). An empty string or empty expression always evaluates to `false`.

---

bool **matchn**(expr: String) const 🔗

Does a simple **case-insensitive** expression match, where `*` matches zero or more arbitrary characters and `?` matches any single character except a period (`.`). An empty string or empty expression always evaluates to `false`.

PackedByteArray **md5_buffer**() const 🔗

Returns the MD5 hash [https://en.wikipedia.org/wiki/MD5] of the string as a PackedByteArray.

String **md5_text**() const 🔗

Returns the MD5 hash [https://en.wikipedia.org/wiki/MD5] of the string as another String.

---

int **naturalcasecmp_to**(to: String) const 🔗

Performs a **case-sensitive**, *natural order* comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "Less than" or "greater than" are determined by the Unicode code points [https://en.wikipedia.org/wiki/List_of_Unicode_characters] of each string, which roughly matches the alphabetical order.

When used for sorting, natural order comparison orders sequences of numbers by the combined value of each digit as is often expected, instead of the single digit's value. A sorted sequence of numbered strings will be `["1", "2", "3", ...]`, not `["1", "10", "2", "3", ...]`.

With different string lengths, returns `1` if this string is longer than the `to` string, or `-1` if shorter. Note that the length of empty strings is *always* `0`.

To get a bool result from a string comparison, use the `==` operator instead. See also naturalnocasecmp_to, filecasecmp_to, and nocasecmp_to.

---

int **naturalnocasecmp_to**(to: String) const 🔗

Performs a **case-insensitive**, *natural order* comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "Less than" or "greater than" are determined by the Unicode code points [https://en.wikipedia.org/wiki/List_of_Unicode_characters] of each string, which roughly matches the alphabetical order. Internally, lowercase characters are converted to uppercase for the comparison.

When used for sorting, natural order comparison orders sequences of numbers by the combined value of each digit as is often expected, instead of the single digit's value. A sorted sequence of numbered strings will be `["1", "2", "3", ...]`, not `["1", "10", "2", "3", ...]`.

With different string lengths, returns `1` if this string is longer than the `to` string, or `-1` if shorter. Note that the length of empty strings is *always* `0`.

To get a bool result from a string comparison, use the `==` operator instead. See also naturalcasecmp_to, filenocasecmp_to, and casecmp_to.

---

int **nocasecmp_to**(to: String) const 🔗

Performs a **case-insensitive** comparison to another string. Returns `-1` if less than, `1` if greater than, or `0` if equal. "Less than" or "greater than" are determined by the Unicode code points [https://en.wikipedia.org/wiki/List_of_Unicode_characters] of each string, which roughly matches the alphabetical order. Internally, lowercase characters are converted to uppercase for the comparison.

With different string lengths, returns `1` if this string is longer than the `to` string, or `-1` if shorter. Note that the length of empty strings is *always* `0`.

To get a bool result from a string comparison, use the `==` operator instead. See also casecmp_to, filenocasecmp_to, and naturalnocasecmp_to.

String **pad_decimals**(digits: int) const

Formats the string representing a number to have an exact number of `digits` *after* the decimal point.

String **pad_zeros**(digits: int) const

Formats the string representing a number to have an exact number of `digits` *before* the decimal point.

String **path_join**(file: String) const

Concatenates `file` at the end of the string as a subpath, adding `/` if necessary.

Example: `"this/is".path_join("path") == "this/is/path"`.

String **repeat**(count: int) const

Repeats this string a number of times. `count` needs to be greater than `0`. Otherwise, returns an empty string.

String **replace**(what: String, forwhat: String) const

Replaces all occurrences of `what` inside the string with the given `forwhat`.

String **replacen**(what: String, forwhat: String) const

Replaces all **case-insensitive** occurrences of `what` inside the string with the given `forwhat`.

String **reverse**() const

Returns the copy of this string in reverse order. This operation works on unicode codepoints, rather than sequences of codepoints, and may break things like compound letters or emojis.

int **rfind**(what: String, from: int = -1) const

Returns the index of the **last** occurrence of `what` in this string, or `-1` if there are none. The search's start can be specified with `from`, continuing to the beginning of the string. This method is the reverse of find.

int **rfindn**(what: String, from: int = -1) const

Returns the index of the **last case-insensitive** occurrence of `what` in this string, or `-1` if there are none. The starting search index can be specified with `from`, continuing to the beginning of the string. This method is the reverse of findn.

String **right**(length: int) const

Returns the last `length` characters from the end of the string. If `length` is negative, strips the first `length` characters from the string's beginning.

```
print("Hello World!".right(3))  # Prints "ld!"
print("Hello World!".right(-4)) # Prints "o World!"
```

String **rpad**(min_length: int, character: String = " ") const

Formats the string to be at least `min_length` long, by adding `character`s to the right of the string, if necessary. See also lpad.

PackedStringArray **rsplit**(delimiter: String = "", allow_empty: bool = true, maxsplit: int = 0) const 🔗

Splits the string using a `delimiter` and returns an array of the substrings, starting from the end of the string. The splits in the returned array appear in the same order as the original string. If `delimiter` is an empty string, each substring will be a single character.

If `allow_empty` is `false`, empty strings between adjacent delimiters are excluded from the array.

If `maxsplit` is greater than `0`, the number of splits may not exceed `maxsplit`. By default, the entire string is split, which is mostly identical to split.

Example:

GDScript

```
var some_string = "One,Two,Three,Four"
var some_array = some_string.rsplit(",", true, 1)

print(some_array.size()) # Prints 2
print(some_array[0])     # Prints "One,Two,Three"
print(some_array[1])     # Prints "Four"
```

C#

```
// In C#, there is no String.RSplit() method.
```

String **rstrip**(chars: String) const 🔗

Removes a set of characters defined in `chars` from the string's end. See also lstrip.

Note: `chars` is not a suffix. Use trim_suffix to remove a single suffix, rather than a set of characters.

PackedByteArray **sha1_buffer**() const 🔗

Returns the SHA-1 [https://en.wikipedia.org/wiki/SHA-1] hash of the string as a PackedByteArray.

String **sha1_text**() const 🔗

Returns the SHA-1 [https://en.wikipedia.org/wiki/SHA-1] hash of the string as another String.

PackedByteArray **sha256_buffer**() const 🔗

Returns the SHA-256 [https://en.wikipedia.org/wiki/SHA-2] hash of the string as a PackedByteArray.

String **sha256_text**() const 🔗

Returns the SHA-256 [https://en.wikipedia.org/wiki/SHA-2] hash of the string as another String.

---

float **similarity**(text: String) const 🔗

Returns the similarity index (Sorensen-Dice coefficient [https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient]) of this string compared to another. A result of `1.0` means totally similar, while `0.0` means totally dissimilar.

```
print("ABC123".similarity("ABC123")) # Prints 1.0
print("ABC123".similarity("XYZ456")) # Prints 0.0
print("ABC123".similarity("123ABC")) # Prints 0.8
print("ABC123".similarity("abc123")) # Prints 0.4
```

String **simplify_path**() const

If the string is a valid file path, converts the string into a canonical path. This is the shortest possible path, without `"./"`, and all the unnecessary `".."` and `"/"`.

```
var simple_path = "./path/to///../file".simplify_path()
print(simple_path) # Prints "path/file"
```

PackedStringArray **split**(delimiter: String = "", allow_empty: bool = true, maxsplit: int = 0) const

Splits the string using a `delimiter` and returns an array of the substrings. If `delimiter` is an empty string, each substring will be a single character. This method is the opposite of join.

If `allow_empty` is `false`, empty strings between adjacent delimiters are excluded from the array.

If `maxsplit` is greater than `0`, the number of splits may not exceed `maxsplit`. By default, the entire string is split.

Example:

GDScript

```
var some_array = "One,Two,Three,Four".split(",", true, 2)

print(some_array.size()) # Prints 3
print(some_array[0])     # Prints "One"
print(some_array[1])      # Prints "Two"
print(some_array[2])      # Prints "Three,Four"
```

C#

```
// C#'s `Split()` does not support the `maxsplit` parameter.
var someArray = "One,Two,Three".Split(",");

GD.Print(someArray[0]); // Prints "One"
GD.Print(someArray[1]); // Prints "Two"
GD.Print(someArray[2]); // Prints "Three"
```

Note: If you only need one substring from the array, consider using get_slice which is faster. If you need to split strings with more complex rules, use the RegEx class instead.

PackedFloat64Array **split_floats**(delimiter: String, allow_empty: bool = true) const 🔗

Splits the string into floats by using a `delimiter` and returns a PackedFloat64Array.

If `allow_empty` is `false`, empty or invalid float conversions between adjacent delimiters are excluded.

```
var a = "1,2,4.5".split_floats(",")         # a is [1.0, 2.0,
4.5]
var c = "1| ||4.5".split_floats("|")        # c is [1.0, 0.0,
0.0, 4.5]
var b = "1| ||4.5".split_floats("|", false) # b is [1.0, 4.5]
```

String **strip_edges**(left: bool = true, right: bool = true) const 🔗

Strips all non-printable characters from the beginning and the end of the string. These include spaces, tabulations (`\t`), and newlines (`\n` `\r`).

If `left` is `false`, ignores the string's beginning. Likewise, if `right` is `false`, ignores the string's end.

String **strip_escapes**() const

Strips all escape characters from the string. These include all non-printable control characters of the first page of the ASCII table (values from 0 to 31), such as tabulation (`\t`) and newline (`\n`, `\r`) characters, but *not* spaces.

String **substr**(from: int, len: int = -1) const

Returns part of the string from the position `from` with length `len`. If `len` is `-1` (as by default), returns the rest of the string starting from the given position.

PackedByteArray **to_ascii_buffer**() const

Converts the string to an ASCII [https://en.wikipedia.org/wiki/ASCII]/Latin-1 encoded PackedByteArray. This method is slightly faster than to_utf8_buffer, but replaces all unsupported characters with spaces.

String **to_camel_case**() const

Returns the string converted to `camelCase`.

float **to_float**() const

Converts the string representing a decimal number into a float. This method stops on the first non-number character, except the first decimal point (`.`) and the exponent letter (`e`). See also is_valid_float.

```
var a = "12.35".to_float() # a is 12.35
var b = "1.2.3".to_float() # b is 1.2
var c = "12xy3".to_float() # c is 12.0
var d = "1e3".to_float()   # d is 1000.0
var e = "Hello!".to_int()  # e is 0.0
```

---

int **to_int**() const

Converts the string representing an integer number into an int. This method removes any non-number character and stops at the first decimal point (`.`). See also is_valid_int.

```
var a = "123".to_int()    # a is 123
var b = "x1y2z3".to_int() # b is 123
var c = "-1.2.3".to_int() # c is -1
var d = "Hello!".to_int() # d is 0
```

String **to_lower**() const

Returns the string converted to `lowercase`.

String **to_pascal_case**() const

Returns the string converted to `PascalCase`.

String **to_snake_case**() const

Returns the string converted to `snake_case`.

Note: Numbers followed by a *single* letter are not separated in the conversion to keep some words (such as "2D") together.

GDScript

```
"Node2D".to_snake_case()               # Returns "node_2d"
"2nd place".to_snake_case()            # Returns "2_nd_place"
"Texture3DAssetFolder".to_snake_case() # Returns
"texture_3d_asset_folder"
```

C#

```
"Node2D".ToSnakeCase();               // Returns "node_2d"
"2nd place".ToSnakeCase();            // Returns "2_nd_place"
"Texture3DAssetFolder".ToSnakeCase(); // Returns
"texture_3d_asset_folder"
```

String **to_upper**() const

Returns the string converted to `UPPERCASE`.

PackedByteArray **to_utf8_buffer**() const

Converts the string to a UTF-8 [https://en.wikipedia.org/wiki/UTF-8] encoded PackedByteArray. This method is slightly slower than to_ascii_buffer, but supports all UTF-8 characters. For most cases, prefer using this method.

PackedByteArray **to_utf16_buffer**() const

Converts the string to a UTF-16 [https://en.wikipedia.org/wiki/UTF-16] encoded PackedByteArray.

PackedByteArray **to_utf32_buffer**() const

Converts the string to a UTF-32 [https://en.wikipedia.org/wiki/UTF-32] encoded PackedByteArray.

PackedByteArray **to_wchar_buffer**() const 🔗

Converts the string to a wide character [https://en.wikipedia.org/wiki/Wide_character] (`wchar_t`, UTF-16 on Windows, UTF-32 on other platforms) encoded PackedByteArray.

String **trim_prefix**(prefix: String) const 🔗

Removes the given `prefix` from the start of the string, or returns the string unchanged.

String **trim_suffix**(suffix: String) const 🔗

Removes the given `suffix` from the end of the string, or returns the string unchanged.

---

int **unicode_at**(at: int) const 🔗

Returns the character code at position `at`.

String **uri_decode**() const 🔗

Decodes the string from its URL-encoded format. This method is meant to properly decode the parameters in a URL when receiving an HTTP request.

GDScript

```
var url = "$DOCS_URL/?highlight=Godot%20Engine%3%docs"
print(url.uri_decode()) # Prints "$DOCS_URL/?highlight=Godot Engine:docs"
```

C#

```
var url = "$DOCS_URL/?highlight=Godot%20Engine%3%docs"
GD.Print(url.URIDecode()) // Prints "$DOCS_URL/?highlight=Godot Engine:docs"
```

String **uri_encode**() const

Encodes the string to URL-friendly format. This method is meant to properly encode the parameters in a URL when sending an HTTP request.

GDScript

```
var prefix = "$DOCS_URL/?highlight="
var url = prefix + "Godot Engine:docs".uri_encode()

print(url) # Prints "$DOCS_URL/?highlight=Godot%20Engine%3%docs"
```

C#

```
var prefix = "$DOCS_URL/?highlight=";
var url = prefix + "Godot Engine:docs".URIEncode();

GD.Print(url); // Prints "$DOCS_URL/?highlight=Godot%20Engine%3%docs"
```

String **validate_filename**() const

Returns a copy of the string with all characters that are not allowed in is_valid_filename replaced with underscores.

String **validate_node_name**() const

Returns a copy of the string with all characters that are not allowed in Node.name (`. : @ / " %`) replaced with underscores.

String **xml_escape**(escape_quotes: bool = false) const

Returns a copy of the string with special characters escaped using the XML standard. If `escape_quotes` is `true`, the single quote (`'`) and double quote (`"`) characters are also escaped.

String **xml_unescape**() const

Returns a copy of the string with escaped characters replaced by their meanings according to the XML standard.

Operator Descriptions

bool **operator !=**(right: String)

Returns `true` if this **StringName** is not equivalent to the given String.

---

bool **operator !=**(right: StringName)

Returns `true` if the **StringName** and `right` do not refer to the same name. Comparisons between **StringName**s are much faster than regular String comparisons.

String **operator %**(right: Variant)

Formats the **StringName**, replacing the placeholders with one or more parameters, returning a String. To pass multiple parameters, `right` needs to be an Array.

For more information, see the GDScript format strings tutorial.

Note: In C#, this operator is not available. Instead, see how to interpolate strings with "$" [https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated].

String **operator +**(right: String)

Appends `right` at the end of this **StringName**, returning a String. This is also known as a string concatenation.

String **operator +**(right: StringName)

Appends `right` at the end of this **StringName**, returning a String. This is also known as a string concatenation.

---

bool **operator <**(right: StringName)

Returns `true` if the left **StringName**'s pointer comes before `right`. Note that this will not match their Unicode order [https://en.wikipedia.org/wiki/List_of_Unicode_characters].

---

bool **operator <=**(right: StringName)

Returns `true` if the left **StringName**'s pointer comes before `right` or if they are the same. Note that this will not match their Unicode order [https://en.wikipedia.org/wiki/List_of_Unicode_characters].

---

bool **operator ==**(right: String)

Returns `true` if this **StringName** is equivalent to the given String.

---

bool **operator ==**(right: StringName)

Returns `true` if the **StringName** and `right` refer to the same name. Comparisons between **StringName**s are much faster than regular String comparisons.

---

bool **operator >**(right: StringName)

Returns `true` if the left **StringName**'s pointer comes after `right`. Note that this will not match their Unicode order [https://en.wikipedia.org/wiki/List_of_Unicode_characters].

---

bool **operator >=**(right: StringName)

Returns `true` if the left **StringName**'s pointer comes after `right` or if they are the same. Note that this will not match their Unicode order [https://en.wikipedia.org/wiki/List_of_Unicode_characters].

Transform2D

A 2×3 matrix representing a 2D transformation.

Description

The **Transform2D** built-in Variant type is a 2×3 matrix [https://en.wikipedia.org/wiki/Matrix_(mathematics)] representing a transformation in 2D space. It contains three Vector2 values: x, y, and origin. Together, they can represent translation, rotation, scale, and skew.

The x and y axes form a 2×2 matrix, known as the transform's **basis**. The length of each axis (Vector2.length) influences the transform's scale, while the direction of all axes influence the rotation. Usually, both axes are perpendicular to one another. However, when you rotate one axis individually, the transform becomes skewed. Applying a skewed transform to a 2D sprite will make the sprite appear distorted.

For a general introduction, see the Matrices and transforms tutorial.

Note: Unlike Transform3D, there is no 2D equivalent to the Basis type. All mentions of "basis" refer to the x and y components of **Transform2D**.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Math documentation index
- Matrices and transforms
- Matrix Transform Demo [https://godotengine.org/asset-library/asset/2787]
- 2.5D Game Demo [https://godotengine.org/asset-library/asset/2783]

Properties

Vector2	origin	`Vector2(0, 0)`
Vector2	x	`Vector2(1, 0)`
Vector2	y	`Vector2(0, 1)`

Constructors

Transform2D	Transform2D()
Transform2D	Transform2D(from: Transform2D)
Transform2D	Transform2D(rotation: float, position: Vector2)
Transform2D	Transform2D(rotation: float, scale: Vector2, skew: float, position: Vector2)

Transform2D	Transform2D(x_axis: Vector2, y_axis: Vector2, origin: Vector2)

Methods

Transform2D	affine_inverse() const
Vector2	basis_xform(v: Vector2) const
Vector2	basis_xform_inv(v: Vector2) const
float	determinant() const
Vector2	get_origin() const
float	get_rotation() const
Vector2	get_scale() const
float	get_skew() const
Transform2D	interpolate_with(xform: Transform2D, weight: float) const

Transform2D	inverse() const
bool	is_conformal() const
bool	is_equal_approx(xform: Transform2D) const
bool	is_finite() const
Transform2D	looking_at(target: Vector2 = Vector2(0, 0)) const
Transform2D	orthonormalized() const
Transform2D	rotated(angle: float) const
Transform2D	rotated_local(angle: float) const
Transform2D	scaled(scale: Vector2) const
Transform2D	scaled_local(scale: Vector2) const
Transform2D	translated(offset: Vector2) const
Transform2D	translated_local(offset: Vector2) const

Operators

bool	operator !=(right: Transform2D)
PackedVector2Array	operator *(right: PackedVector2Array)
Rect2	operator *(right: Rect2)
Transform2D	operator *(right: Transform2D)
Vector2	operator *(right: Vector2)
Transform2D	operator *(right: float)
Transform2D	operator *(right: int)
Transform2D	operator /(right: float)
Transform2D	operator /(right: int)
bool	operator ==(right: Transform2D)
Vector2	operator [](index: int)

Constants

IDENTITY = `Transform2D(1, 0, 0, 1, 0, 0)` 🔗

The identity **Transform2D**. A transform with no translation, no rotation, and its scale being `1`. When multiplied by another Variant such as Rect2 or another **Transform2D**, no transformation occurs. This means that:

- The x points right (Vector2.RIGHT);
- The y points up (Vector2.UP).

```
var transform = Transform2D.IDENTITY
print("| X | Y | Origin")
print("| %s | %s | %s" % [transform.x.x, transform.y.x, transform.origin.x])
print("| %s | %s | %s" % [transform.x.y, transform.y.y, transform.origin.y])
# Prints:
# | X | Y | Origin
# | 1 | 0 | 0
# | 0 | 1 | 0
```

This is identical to creating Transform2D without any parameters. This constant can be used to make your code clearer, and for consistency with C#.

FLIP_X = `Transform2D(-1, 0, 0, 1, 0, 0)` 🔗

When any transform is multiplied by FLIP_X, it negates all components of the x axis (the X column).

When FLIP_X is multiplied by any basis, it negates the Vector2.x component of all axes (the X row).

FLIP_Y = `Transform2D(1, 0, 0, -1, 0, 0)` 🔗

When any transform is multiplied by FLIP_Y, it negates all components of the y axis (the Y column).

When FLIP_Y is multiplied by any basis, it negates the Vector2.y component of all axes (the Y row).

Property Descriptions

Vector2 **origin** = `Vector2(0, 0)`

The translation offset of this transform, and the column `2` of the matrix. In 2D space, this can be seen as the position.

Vector2 **x** = `Vector2(1, 0)`

The transform basis's X axis, and the column `0` of the matrix. Combined with y, this represents the transform's rotation, scale, and skew.

On the identity transform, this vector points right (Vector2.RIGHT).

Vector2 **y** = `Vector2(0, 1)`

The transform basis's Y axis, and the column `1` of the matrix. Combined with x, this represents the transform's rotation, scale, and skew.

On the identity transform, this vector points up (Vector2.UP).

Constructor Descriptions

Transform2D **Transform2D**() 🔗

Constructs a **Transform2D** identical to IDENTITY.

Transform2D **Transform2D**(from: Transform2D)

Constructs a **Transform2D** as a copy of the given **Transform2D**.

Transform2D **Transform2D**(rotation: float, position: Vector2)

Constructs a **Transform2D** from a given angle (in radians) and position.

Transform2D **Transform2D**(rotation: float, scale: Vector2, skew: float, position: Vector2)

Constructs a **Transform2D** from a given angle (in radians), scale, skew (in radians), and position.

Transform2D **Transform2D**(x_axis: Vector2, y_axis: Vector2, origin: Vector2)

Constructs a **Transform2D** from 3 Vector2 values representing x, y, and the origin (the three matrix columns).

Method Descriptions

Transform2D **affine_inverse**() const 🔗

Returns the inverted version of this transform. Unlike inverse, this method works with almost any basis, including non-uniform ones, but is slower. See also inverse.

Note: For this method to return correctly, the transform's basis needs to have a determinant that is not exactly `0` (see determinant).

Vector2 **basis_xform**(v: Vector2) const 🔗

Returns a copy of the `v` vector, transformed (multiplied) by the transform basis's matrix. Unlike the multiplication operator (`*`), this method ignores the origin.

Vector2 **basis_xform_inv**(v: Vector2) const 🔗

Returns a copy of the `v` vector, transformed (multiplied) by the inverse transform basis's matrix (see inverse). This method ignores the origin.

Note: This method assumes that this transform's basis is *orthonormal* (see orthonormalized). If the basis is not orthonormal, `transform.affine_inverse().basis_xform(vector)` should be used instead (see affine_inverse).

---

float **determinant**() const 🔗

Returns the determinant [https://en.wikipedia.org/wiki/Determinant] of this transform basis's matrix. For advanced math, this number can be used to determine a few attributes:

- If the determinant is exactly `0`, the basis is not invertible (see inverse).
- If the determinant is a negative number, the basis represents a negative scale.

Note: If the basis's scale is the same for every axis, its determinant is always that scale by the power of 2.

Vector2 **get_origin**() const 🔗

Returns this transform's translation. Equivalent to origin.

---

float **get_rotation**() const 🔗

Returns this transform's rotation (in radians). This is equivalent to x's angle (see Vector2.angle).

Vector2 **get_scale**() const 🔗

Returns the length of both x and y, as a Vector2. If this transform's basis is not skewed, this value is the scaling factor. It is not affected by rotation.

GDScript

```
var my_transform = Transform2D(
    Vector2(2, 0),
    Vector2(0, 4),
    Vector2(0, 0)
)
# Rotating the Transform2D in any way preserves its scale.
my_transform = my_transform.rotated(TAU / 2)

print(my_transform.get_scale()) # Prints (2, 4).
```

C#

```
var myTransform = new Transform2D(
    Vector3(2.0f, 0.0f),
    Vector3(0.0f, 4.0f),
    Vector3(0.0f, 0.0f)
);
// Rotating the Transform2D in any way preserves its scale.
myTransform = myTransform.Rotated(Mathf.Tau / 2.0f);

GD.Print(myTransform.GetScale()); // Prints (2, 4, 8).
```

Note: If the value returned by determinant is negative, the scale is also negative.

---

float **get_skew**() const

Returns this transform's skew (in radians).

Transform2D **interpolate_with**(xform: Transform2D, weight: float) const

Returns the result of the linear interpolation between this transform and `xform` by the given `weight`.

The `weight` should be between `0.0` and `1.0` (inclusive). Values outside this range are allowed and can be used to perform *extrapolation* instead.

Transform2D **inverse**() const

Returns the inverted version of this transform [https://en.wikipedia.org/wiki/Invertible_matrix].

Note: For this method to return correctly, the transform's basis needs to be *orthonormal* (see orthonormalized). That means, the basis should only represent a rotation. If it does not, use affine_inverse instead.

---

bool **is_conformal**() const

Returns `true` if this transform's basis is conformal. A conformal basis is both *orthogonal* (the axes are perpendicular to each other) and *uniform* (the axes share the same length). This method can be especially useful during physics calculations.

---

bool **is_equal_approx**(xform: Transform2D) const

Returns `true` if this transform and `xform` are approximately equal, by running @GlobalScope.is_equal_approx on each component.

---

bool **is_finite**() const

Returns `true` if this transform is finite, by calling @GlobalScope.is_finite on each component.

Transform2D **looking_at**(target: Vector2 = Vector2(0, 0)) const

Returns a copy of the transform rotated such that the rotated X-axis points towards the `target` position, in global space.

Transform2D **orthonormalized**() const

Returns a copy of this transform with its basis orthonormalized. An orthonormal basis is both *orthogonal* (the axes are perpendicular to each other) and *normalized* (the axes have a length of `1`), which also means it can only represent rotation.

Transform2D **rotated**(angle: float) const

Returns a copy of the transform rotated by the given `angle` (in radians).

This method is an optimized version of multiplying the given transform `X` with a corresponding rotation transform `R` from the left, i.e., `R * X`.

This can be seen as transforming with respect to the global/parent frame.

Transform2D **rotated_local**(angle: float) const

Returns a copy of the transform rotated by the given `angle` (in radians).

This method is an optimized version of multiplying the given transform `X` with a corresponding rotation transform `R` from the right, i.e., `X * R`.

This can be seen as transforming with respect to the local frame.

Transform2D **scaled**(scale: Vector2) const

Returns a copy of the transform scaled by the given `scale` factor.

This method is an optimized version of multiplying the given transform `X` with a corresponding scaling transform `S` from the left, i.e., `S * X`.

This can be seen as transforming with respect to the global/parent frame.

Transform2D **scaled_local**(scale: Vector2) const

Returns a copy of the transform scaled by the given `scale` factor.

This method is an optimized version of multiplying the given transform `X` with a corresponding scaling transform `S` from the right, i.e., `X * S`.

This can be seen as transforming with respect to the local frame.

Transform2D **translated**(offset: Vector2) const 🔗

Returns a copy of the transform translated by the given `offset`.

This method is an optimized version of multiplying the given transform `X` with a corresponding translation transform `T` from the left, i.e., `T * X`.

This can be seen as transforming with respect to the global/parent frame.

Transform2D **translated_local**(offset: Vector2) const 🔗

Returns a copy of the transform translated by the given `offset`.

This method is an optimized version of multiplying the given transform `X` with a corresponding translation transform `T` from the right, i.e., `X * T`.

This can be seen as transforming with respect to the local frame.

Operator Descriptions

bool **operator !=**(right: Transform2D) 🔗

Returns `true` if the components of both transforms are not equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

PackedVector2Array **operator** *(right: PackedVector2Array) 🔗

Transforms (multiplies) every Vector2 element of the given PackedVector2Array by this transformation matrix.

On larger arrays, this operation is much faster than transforming each Vector2 individually.

Rect2 **operator** *(right: Rect2) 🔗

Transforms (multiplies) the Rect2 by this transformation matrix.

Transform2D **operator** *(right: Transform2D) 🔗

Transforms (multiplies) this transform by the `right` transform.

This is the operation performed between parent and child CanvasItem nodes.

Note: If you need to only modify one attribute of this transform, consider using one of the following methods, instead:

- For translation, see translated or translated_local.
- For rotation, see rotated or rotated_local.
- For scale, see scaled or scaled_local.

Vector2 **operator** *(right: Vector2) 🔗

Transforms (multiplies) the Vector2 by this transformation matrix.

Transform2D **operator ***(right: float)

Multiplies all components of the **Transform2D** by the given float, including the origin. This affects the transform's scale uniformly.

Transform2D **operator ***(right: int)

Multiplies all components of the **Transform2D** by the given int, including the origin. This affects the transform's scale uniformly.

Transform2D **operator /**(right: float)

Divides all components of the **Transform2D** by the given float, including the origin. This affects the transform's scale uniformly.

Transform2D **operator /**(right: int)

Divides all components of the **Transform2D** by the given int, including the origin. This affects the transform's scale uniformly.

---

bool **operator ==**(right: Transform2D)

Returns `true` if the components of both transforms are exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Vector2 **operator []**(index: int)

Accesses each axis (column) of this transform by their index. Index `0` is the same as x, index `1` is the same as y, and index `2` is the same as origin.

Transform3D

A 3×4 matrix representing a 3D transformation.

Description

The **Transform3D** built-in Variant type is a 3×4 matrix representing a transformation in 3D space. It contains a Basis, which on its own can represent rotation, scale, and shear. Additionally, combined with its own origin, the transform can also represent a translation.

For a general introduction, see the Matrices and transforms tutorial.

Note: Godot uses a right-handed coordinate system [https://en.wikipedia.org/wiki/Right-hand_rule], which is a common standard. For directions, the convention for built-in types like Camera3D is for -Z to point forward (+X is right, +Y is up, and +Z is back). Other objects may use different direction conventions. For more information, see the 3D asset direction conventions tutorial.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Math documentation index
- Matrices and transforms
- Using 3D transforms

- Matrix Transform Demo [https://godotengine.org/asset-library/asset/2787]
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- 2.5D Game Demo [https://godotengine.org/asset-library/asset/2783]

Properties

Basis	basis	`Basis(1, 0, 0, 0, 1, 0, 0, 0, 1)`
Vector3	origin	`Vector3(0, 0, 0)`

Constructors

Transform3D	Transform3D()
Transform3D	Transform3D(from: Transform3D)
Transform3D	Transform3D(basis: Basis, origin: Vector3)
Transform3D	Transform3D(from: Projection)
Transform3D	Transform3D(x_axis: Vector3, y_axis: Vector3, z_axis: Vector3, origin: Vector3)

Methods

Transform3D	affine_inverse() const
Transform3D	interpolate_with(xform: Transform3D, weight: float) const
Transform3D	inverse() const
bool	is_equal_approx(xform: Transform3D) const
bool	is_finite() const
Transform3D	looking_at(target: Vector3, up: Vector3 = Vector3(0, 1, 0), use_model_front: bool = false) const
Transform3D	orthonormalized() const
Transform3D	rotated(axis: Vector3, angle: float) const
Transform3D	rotated_local(axis: Vector3, angle: float) const
Transform3D	scaled(scale: Vector3) const
Transform3D	scaled_local(scale: Vector3) const
Transform3D	translated(offset: Vector3) const

Transform3D	translated_local(offset: Vector3) const

Operators

bool	operator !=(right: Transform3D)
AABB	operator *(right: AABB)
PackedVector3Array	operator *(right: PackedVector3Array)
Plane	operator *(right: Plane)
Transform3D	operator *(right: Transform3D)
Vector3	operator *(right: Vector3)
Transform3D	operator *(right: float)
Transform3D	operator *(right: int)
Transform3D	operator /(right: float)
Transform3D	operator /(right: int)

bool	operator ==(right: Transform3D)

Constants

IDENTITY = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`

A transform with no translation, no rotation, and its scale being `1`. Its basis is equal to Basis.IDENTITY.

When multiplied by another Variant such as AABB or another **Transform3D**, no transformation occurs.

FLIP_X = `Transform3D(-1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`

Transform3D with mirroring applied perpendicular to the YZ plane. Its basis is equal to Basis.FLIP_X.

FLIP_Y = `Transform3D(1, 0, 0, 0, -1, 0, 0, 0, 1, 0, 0, 0)`

Transform3D with mirroring applied perpendicular to the XZ plane. Its basis is equal to Basis.FLIP_Y.

FLIP_Z = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, -1, 0, 0, 0)`

Transform3D with mirroring applied perpendicular to the XY plane. Its basis is equal to Basis.FLIP_Z.

Property Descriptions

Basis **basis** = `Basis(1, 0, 0, 0, 1, 0, 0, 0, 1)`

The Basis of this transform. It is composed by 3 axes (Basis.x, Basis.y, and Basis.z). Together, these represent the transform's rotation, scale, and shear.

Vector3 **origin** = `Vector3(0, 0, 0)` 🔗

The translation offset of this transform. In 3D space, this can be seen as the position.

Constructor Descriptions

Transform3D **Transform3D**() 🔗

Constructs a **Transform3D** identical to the IDENTITY.

Transform3D **Transform3D**(from: Transform3D)

Constructs a **Transform3D** as a copy of the given **Transform3D**.

Transform3D **Transform3D**(basis: Basis, origin: Vector3)

Constructs a **Transform3D** from a Basis and Vector3.

Transform3D **Transform3D**(from: Projection)

Constructs a **Transform3D** from a Projection. Because **Transform3D** is a 3×4 matrix and Projection is a 4×4 matrix, this operation trims the last row of the projection matrix (`from.x.w`, `from.y.w`, `from.z.w`, and `from.w.w` are not included in the new transform).

Transform3D **Transform3D**(x_axis: Vector3, y_axis: Vector3, z_axis: Vector3, origin: Vector3)

Constructs a **Transform3D** from four Vector3 values (also called matrix columns).

The first three arguments are the basis's axes (Basis.x, Basis.y, and Basis.z).

Method Descriptions

Transform3D **affine_inverse**() const 🔗

Returns the inverted version of this transform. Unlike inverse, this method works with almost any basis, including non-uniform ones, but is slower. See also Basis.inverse.

Note: For this method to return correctly, the transform's basis needs to have a determinant that is not exactly `0` (see Basis.determinant).

Transform3D **interpolate_with**(xform: Transform3D, weight: float) const 🔗

Returns the result of the linear interpolation between this transform and `xform` by the given `weight`.

The `weight` should be between `0.0` and `1.0` (inclusive). Values outside this range are allowed and can be used to perform *extrapolation* instead.

Transform3D **inverse**() const 🔗

Returns the inverted version of this transform. See also Basis.inverse.

Note: For this method to return correctly, the transform's basis needs to be *orthonormal* (see Basis.orthonormalized). That means, the basis should only represent a rotation. If it does not, use affine_inverse instead.

bool **is_equal_approx**(xform: Transform3D) const 🔗

Returns `true` if this transform and `xform` are approximately equal, by running @GlobalScope.is_equal_approx on each component.

bool **is_finite**() const 🔗

Returns `true` if this transform is finite, by calling @GlobalScope.is_finite on each component.

Transform3D **looking_at**(target: Vector3, up: Vector3 = Vector3(0, 1, 0), use_model_front: bool = false) const 🔗

Returns a copy of this transform rotated so that the forward axis (-Z) points towards the `target` position.

The up axis (+Y) points as close to the `up` vector as possible while staying perpendicular to the forward axis. The resulting transform is orthonormalized. The existing rotation, scale, and skew information from the original transform is discarded. The `target` and `up` vectors cannot be zero, cannot be parallel to each other, and are defined in global/parent space.

If `use_model_front` is `true`, the +Z axis (asset front) is treated as forward (implies +X is left) and points toward the `target` position. By default, the -Z axis (camera forward) is treated as forward (implies +X is right).

Transform3D **orthonormalized**() const

Returns a copy of this transform with its basis orthonormalized. An orthonormal basis is both *orthogonal* (the axes are perpendicular to each other) and *normalized* (the axes have a length of `1`), which also means it can only represent rotation. See also Basis.orthonormalized.

Transform3D **rotated**(axis: Vector3, angle: float) const

Returns a copy of this transform rotated around the given `axis` by the given `angle` (in radians).

The `axis` must be a normalized vector.

This method is an optimized version of multiplying the given transform `X` with a corresponding rotation transform `R` from the left, i.e., `R * X`.

This can be seen as transforming with respect to the global/parent frame.

Transform3D **rotated_local**(axis: Vector3, angle: float) const

Returns a copy of this transform rotated around the given `axis` by the given `angle` (in radians).

The `axis` must be a normalized vector.

This method is an optimized version of multiplying the given transform `X` with a corresponding rotation transform `R` from the right, i.e., `X * R`.

This can be seen as transforming with respect to the local frame.

Transform3D **scaled**(scale: Vector3) const

Returns a copy of this transform scaled by the given `scale` factor.

This method is an optimized version of multiplying the given transform `X` with a corresponding scaling transform `S` from the left, i.e., `S * X`.

This can be seen as transforming with respect to the global/parent frame.

Transform3D **scaled_local**(scale: Vector3) const

Returns a copy of this transform scaled by the given `scale` factor.

This method is an optimized version of multiplying the given transform `X` with a corresponding scaling transform `S` from the right, i.e., `X * S`.

This can be seen as transforming with respect to the local frame.

Transform3D **translated**(offset: Vector3) const

Returns a copy of this transform translated by the given `offset`.

This method is an optimized version of multiplying the given transform `X` with a corresponding translation transform `T` from the left, i.e., `T * X`.

This can be seen as transforming with respect to the global/parent frame.

Transform3D **translated_local**(offset: Vector3) const

Returns a copy of this transform translated by the given `offset`.

This method is an optimized version of multiplying the given transform `X` with a corresponding translation transform `T` from the right, i.e., `X * T`.

This can be seen as transforming with respect to the local frame.

Operator Descriptions

bool **operator !=**(right: Transform3D)

Returns `true` if the components of both transforms are not equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

AABB **operator ***(right: AABB)

Transforms (multiplies) the AABB by this transformation matrix.

PackedVector3Array **operator ***(right: PackedVector3Array)

Transforms (multiplies) every Vector3 element of the given PackedVector3Array by this transformation matrix.

On larger arrays, this operation is much faster than transforming each Vector3 individually.

Plane **operator ***(right: Plane)

Transforms (multiplies) the Plane by this transformation matrix.

Transform3D **operator ***(right: Transform3D)

Transforms (multiplies) this transform by the `right` transform.

This is the operation performed between parent and child Node3Ds.

Note: If you need to only modify one attribute of this transform, consider using one of the following methods, instead:

- For translation, see translated or translated_local.
- For rotation, see rotated or rotated_local.
- For scale, see scaled or scaled_local.

Vector3 **operator ***(right: Vector3)

Transforms (multiplies) the Vector3 by this transformation matrix.

Transform3D **operator ***(right: float)

Multiplies all components of the **Transform3D** by the given float, including the origin. This affects the transform's scale uniformly, scaling the basis.

Transform3D **operator ***(right: int)

Multiplies all components of the **Transform3D** by the given int, including the origin. This affects the transform's scale uniformly, scaling the basis.

Transform3D **operator /**(right: float)

Divides all components of the **Transform3D** by the given float, including the origin. This affects the transform's scale uniformly, scaling the basis.

Transform3D **operator /**(right: int)

Divides all components of the **Transform3D** by the given int, including the origin. This affects the transform's scale uniformly, scaling the basis.

---

bool **operator ==**(right: Transform3D)

Returns `true` if the components of both transforms are exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Vector2

A 2D vector using floating-point coordinates.

Description

A 2-element structure that can be used to represent 2D coordinates or any other pair of numeric values.

It uses floating-point coordinates. By default, these floating-point values use 32-bit precision, unlike float which is always 64-bit. If double precision is needed, compile the engine with the option `precision=double`.

See Vector2i for its integer counterpart.

Note: In a boolean context, a Vector2 will evaluate to `false` if it's equal to `Vector2(0, 0)`. Otherwise, a Vector2 will always evaluate to `true`.

Tutorials

- Math documentation index
- Vector math
- Advanced vector math
- 3Blue1Brown Essence of Linear Algebra [https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab]
- Matrix Transform Demo [https://godotengine.org/asset-library/asset/2787]
- All 2D Demos [https://github.com/godotengine/godot-demo-projects/tree/master/2d]

Properties

float	x	`0.0`
float	y	`0.0`

Constructors

Vector2	Vector2()
Vector2	Vector2(from: Vector2)
Vector2	Vector2(from: Vector2i)
Vector2	Vector2(x: float, y: float)

Methods

Vector2	abs() const
float	angle() const
float	angle_to(to: Vector2) const

float	angle_to_point(to: Vector2) const
float	aspect() const
Vector2	bezier_derivative(control_1: Vector2, control_2: Vector2, end: Vector2, t: float) const
Vector2	bezier_interpolate(control_1: Vector2, control_2: Vector2, end: Vector2, t: float) const
Vector2	bounce(n: Vector2) const
Vector2	ceil() const
Vector2	clamp(min: Vector2, max: Vector2) const
Vector2	clampf(min: float, max: float) const
float	cross(with: Vector2) const
Vector2	cubic_interpolate(b: Vector2, pre_a: Vector2, post_b: Vector2, weight: float) const
Vector2	cubic_interpolate_in_time(b: Vector2, pre_a: Vector2, post_b: Vector2, weight: float, b_t: float, pre_a_t: float, post_b_t: float) const

Vector2	direction_to(to: Vector2) const
float	distance_squared_to(to: Vector2) const
float	distance_to(to: Vector2) const
float	dot(with: Vector2) const
Vector2	floor() const
Vector2	from_angle(angle: float) static
bool	is_equal_approx(to: Vector2) const
bool	is_finite() const
bool	is_normalized() const
bool	is_zero_approx() const
float	length() const
float	length_squared() const
Vector2	lerp(to: Vector2, weight: float) const

Vector2	limit_length(length: float = 1.0) const
Vector2	max(with: Vector2) const
int	max_axis_index() const
Vector2	maxf(with: float) const
Vector2	min(with: Vector2) const
int	min_axis_index() const
Vector2	minf(with: float) const
Vector2	move_toward(to: Vector2, delta: float) const
Vector2	normalized() const
Vector2	orthogonal() const
Vector2	posmod(mod: float) const
Vector2	posmodv(modv: Vector2) const
Vector2	project(b: Vector2) const

Vector2	reflect(line: Vector2) const
Vector2	rotated(angle: float) const
Vector2	round() const
Vector2	sign() const
Vector2	slerp(to: Vector2, weight: float) const
Vector2	slide(n: Vector2) const
Vector2	snapped(step: Vector2) const
Vector2	snappedf(step: float) const

Operators

bool	operator !=(right: Vector2)
Vector2	operator *(right: Transform2D)
Vector2	operator *(right: Vector2)

Vector2	operator *(right: float)
Vector2	operator *(right: int)
Vector2	operator +(right: Vector2)
Vector2	operator -(right: Vector2)
Vector2	operator /(right: Vector2)
Vector2	operator /(right: float)
Vector2	operator /(right: int)
bool	operator <(right: Vector2)
bool	operator <=(right: Vector2)
bool	operator ==(right: Vector2)
bool	operator >(right: Vector2)
bool	operator >=(right: Vector2)
float	operator [](index: int)

Vector2	operator unary+()
Vector2	operator unary-()

Constants

AXIS_X = `0`

Enumerated value for the X axis. Returned by max_axis_index and min_axis_index.

AXIS_Y = `1`

Enumerated value for the Y axis. Returned by max_axis_index and min_axis_index.

ZERO = `Vector2(0, 0)`

Zero vector, a vector with all components set to `0`.

ONE = `Vector2(1, 1)`

One vector, a vector with all components set to `1`.

INF = `Vector2(inf, inf)`

Infinity vector, a vector with all components set to @GDScript.INF.

LEFT = `Vector2(-1, 0)`

Left unit vector. Represents the direction of left.

RIGHT = `Vector2(1, 0)`

Right unit vector. Represents the direction of right.

UP = `Vector2(0, -1)`

Up unit vector. Y is down in 2D, so this vector points -Y.

DOWN = `Vector2(0, 1)`

Down unit vector. Y is down in 2D, so this vector points +Y.

Property Descriptions

float **x** = `0.0`

The vector's X component. Also accessible by using the index position `[0]`.

---

float **y** = `0.0`

The vector's Y component. Also accessible by using the index position `[1]`.

Constructor Descriptions

Vector2 **Vector2**()

Constructs a default-initialized **Vector2** with all components set to `0`.

Vector2 **Vector2**(from: Vector2)

Constructs a **Vector2** as a copy of the given **Vector2**.

Vector2 **Vector2**(from: Vector2i)

Constructs a new **Vector2** from Vector2i.

Vector2 **Vector2**(x: float, y: float)

Constructs a new **Vector2** from the given `x` and `y`.

Method Descriptions

Vector2 **abs**() const 🔗

Returns a new vector with all components in absolute values (i.e. positive).

---

float **angle**() const 🔗

Returns this vector's angle with respect to the positive X axis, or `(1, 0)` vector, in radians.

For example, `Vector2.RIGHT.angle()` will return zero, `Vector2.DOWN.angle()` will return `PI / 2` (a quarter turn, or 90 degrees), and `Vector2(1, -1).angle()` will return `-PI / 4` (a negative eighth turn, or -45 degrees).

Illustration of the returned angle.
[https://raw.githubusercontent.com/godotengine/godot-docs/master/img/vector2_angle.png]

Equivalent to the result of @GlobalScope.atan2 when called with the vector's y and x as parameters: `atan2(y, x)`.

---

float **angle_to**(to: Vector2) const

Returns the angle to the given vector, in radians.

Illustration of the returned angle. [https://raw.githubusercontent.com/godotengine/godot-docs/master/img/vector2_angle_to.png]

---

float **angle_to_point**(to: Vector2) const

Returns the angle between the line connecting the two points and the X axis, in radians.

`a.angle_to_point(b)` is equivalent of doing `(b - a).angle()`.

Illustration of the returned angle. [https://raw.githubusercontent.com/godotengine/godot-docs/master/img/vector2_angle_to_point.png]

---

float **aspect**() const

Returns the aspect ratio of this vector, the ratio of x to y.

Vector2 **bezier_derivative**(control_1: Vector2, control_2: Vector2, end: Vector2, t: float) const

Returns the derivative at the given `t` on the Bézier curve [https://en.wikipedia.org/wiki/B%C3%A9zier_curve] defined by this vector and the given `control_1`, `control_2`, and `end` points.

Vector2 **bezier_interpolate**(control_1: Vector2, control_2: Vector2, end: Vector2, t: float) const 🔗

Returns the point at the given `t` on the Bézier curve [https://en.wikipedia.org/wiki/B%C3%A9zier_curve] defined by this vector and the given `control_1`, `control_2`, and `end` points.

Vector2 **bounce**(n: Vector2) const 🔗

Returns the vector "bounced off" from a line defined by the given normal `n` perpendicular to the line.

Note: bounce performs the operation that most engines and frameworks call `reflect()`.

Vector2 **ceil**() const 🔗

Returns a new vector with all components rounded up (towards positive infinity).

Vector2 **clamp**(min: Vector2, max: Vector2) const 🔗

Returns a new vector with all components clamped between the components of `min` and `max`, by running @GlobalScope.clamp on each component.

Vector2 **clampf**(min: float, max: float) const 🔗

Returns a new vector with all components clamped between `min` and `max`, by running @GlobalScope.clamp on each component.

---

float **cross**(with: Vector2) const

Returns the 2D analog of the cross product for this vector and `with`.

This is the signed area of the parallelogram formed by the two vectors. If the second vector is clockwise from the first vector, then the cross product is the positive area. If counter-clockwise, the cross product is the negative area. If the two vectors are parallel this returns zero, making it useful for testing if two vectors are parallel.

Note: Cross product is not defined in 2D mathematically. This method embeds the 2D vectors in the XY plane of 3D space and uses their cross product's Z component as the analog.

Vector2 **cubic_interpolate**(b: Vector2, pre_a: Vector2, post_b: Vector2, weight: float) const

Performs a cubic interpolation between this vector and `b` using `pre_a` and `post_b` as handles, and returns the result at position `weight`. `weight` is on the range of 0.0 to 1.0, representing the amount of interpolation.

Vector2 **cubic_interpolate_in_time**(b: Vector2, pre_a: Vector2, post_b: Vector2, weight: float, b_t: float, pre_a_t: float, post_b_t: float) const

Performs a cubic interpolation between this vector and `b` using `pre_a` and `post_b` as handles, and returns the result at position `weight`. `weight` is on the range of 0.0 to 1.0, representing the amount of interpolation.

It can perform smoother interpolation than cubic_interpolate by the time values.

Vector2 **direction_to**(to: Vector2) const 🔗

Returns the normalized vector pointing from this vector to `to`. This is equivalent to using `(b - a).normalized()`.

---

float **distance_squared_to**(to: Vector2) const 🔗

Returns the squared distance between this vector and `to`.

This method runs faster than distance_to, so prefer it if you need to compare vectors or need the squared distance for some formula.

---

float **distance_to**(to: Vector2) const 🔗

Returns the distance between this vector and `to`.

---

float **dot**(with: Vector2) const 🔗

Returns the dot product of this vector and `with`. This can be used to compare the angle between two vectors. For example, this can be used to determine whether an enemy is facing the player.

The dot product will be `0` for a right angle (90 degrees), greater than 0 for angles narrower than 90 degrees and lower than 0 for angles wider than 90 degrees.

When using unit (normalized) vectors, the result will always be between `-1.0` (180 degree angle) when the vectors are facing opposite directions, and `1.0` (0 degree angle) when the vectors are aligned.

Note: `a.dot(b)` is equivalent to `b.dot(a)`.

Vector2 **floor**() const

Returns a new vector with all components rounded down (towards negative infinity).

Vector2 **from_angle**(angle: float) static

Creates a unit **Vector2** rotated to the given `angle` in radians. This is equivalent to doing `Vector2(cos(angle), sin(angle))` or `Vector2.RIGHT.rotated(angle)`.

```
print(Vector2.from_angle(0)) # Prints (1, 0).
print(Vector2(1, 0).angle()) # Prints 0, which is the angle used above.
print(Vector2.from_angle(PI / 2)) # Prints (0, 1).
```

bool **is_equal_approx**(to: Vector2) const

Returns `true` if this vector and `to` are approximately equal, by running @GlobalScope.is_equal_approx on each component.

bool **is_finite**() const

Returns `true` if this vector is finite, by calling @GlobalScope.is_finite on each component.

bool **is_normalized**() const

Returns `true` if the vector is normalized, i.e. its length is approximately equal to 1.

bool **is_zero_approx**() const

Returns `true` if this vector's values are approximately zero, by running @GlobalScope.is_zero_approx on each component.

This method is faster than using is_equal_approx with one value as a zero vector.

---

float **length**() const

Returns the length (magnitude) of this vector.

---

float **length_squared**() const

Returns the squared length (squared magnitude) of this vector.

This method runs faster than length, so prefer it if you need to compare vectors or need the squared distance for some formula.

Vector2 **lerp**(to: Vector2, weight: float) const

Returns the result of the linear interpolation between this vector and `to` by amount `weight`. `weight` is on the range of `0.0` to `1.0`, representing the amount of interpolation.

Vector2 **limit_length**(length: float = 1.0) const

Returns the vector with a maximum length by limiting its length to `length`.

Vector2 **max**(with: Vector2) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector2(maxf(x, with.x), maxf(y, with.y))`.

int **max_axis_index**() const

Returns the axis of the vector's highest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_X.

Vector2 **maxf**(with: float) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector2(maxf(x, with), maxf(y, with))`.

Vector2 **min**(with: Vector2) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector2(minf(x, with.x), minf(y, with.y))`.

int **min_axis_index**() const

Returns the axis of the vector's lowest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_Y.

Vector2 **minf**(with: float) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector2(minf(x, with), minf(y, with))`.

Vector2 **move_toward**(to: Vector2, delta: float) const

Returns a new vector moved toward `to` by the fixed `delta` amount. Will not go past the final value.

Vector2 **normalized**() const

Returns the result of scaling the vector to unit length. Equivalent to `v / v.length()`. Returns `(0, 0)` if `v.length() == 0`. See also is_normalized.

Note: This function may return incorrect values if the input vector length is near zero.

Vector2 **orthogonal**() const

Returns a perpendicular vector rotated 90 degrees counter-clockwise compared to the original, with the same length.

Vector2 **posmod**(mod: float) const

Returns a vector composed of the @GlobalScope.fposmod of this vector's components and `mod`.

Vector2 **posmodv**(modv: Vector2) const

Returns a vector composed of the @GlobalScope.fposmod of this vector's components and `modv`'s components.

Vector2 **project**(b: Vector2) const

Returns a new vector resulting from projecting this vector onto the given vector `b`. The resulting new vector is parallel to `b`. See also slide.

Note: If the vector `b` is a zero vector, the components of the resulting new vector will be @GDScript.NAN.

Vector2 **reflect**(line: Vector2) const

Returns the result of reflecting the vector from a line defined by the given direction vector `line`.

Note: reflect differs from what other engines and frameworks call `reflect()`. In other engines, `reflect()` takes a normal direction which is a direction perpendicular to the line. In Godot, you specify the direction of the line directly. See also bounce which does what most engines call `reflect()`.

Vector2 **rotated**(angle: float) const

Returns the result of rotating this vector by `angle` (in radians). See also @GlobalScope.deg_to_rad.

Vector2 **round**() const

Returns a new vector with all components rounded to the nearest integer, with halfway cases rounded away from zero.

Vector2 **sign**() const

Returns a new vector with each component set to `1.0` if it's positive, `-1.0` if it's negative, and `0.0` if it's zero. The result is identical to calling @GlobalScope.sign on each component.

Vector2 **slerp**(to: Vector2, weight: float) const

Returns the result of spherical linear interpolation between this vector and `to`, by amount `weight`. `weight` is on the range of 0.0 to

1.0, representing the amount of interpolation.

This method also handles interpolating the lengths if the input vectors have different lengths. For the special case of one or both input vectors having zero length, this method behaves like lerp.

Vector2 **slide**(n: Vector2) const

Returns a new vector resulting from sliding this vector along a line with normal `n`. The resulting new vector is perpendicular to `n`, and is equivalent to this vector minus its projection on `n`. See also project.

Note: The vector `n` must be normalized. See also normalized.

Vector2 **snapped**(step: Vector2) const

Returns a new vector with each component snapped to the nearest multiple of the corresponding component in `step`. This can also be used to round the components to an arbitrary number of decimals.

Vector2 **snappedf**(step: float) const

Returns a new vector with each component snapped to the nearest multiple of `step`. This can also be used to round the components to an arbitrary number of decimals.

Operator Descriptions

bool **operator !=**(right: Vector2)

Returns `true` if the vectors are not equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

Vector2 **operator ***(right: Transform2D) 🔗

Inversely transforms (multiplies) the **Vector2** by the given Transform2D transformation matrix, under the assumption that the transformation basis is orthonormal (i.e. rotation/reflection is fine, scaling/skew is not).

`vector * transform` is equivalent to `transform.inverse() * vector`. See Transform2D.inverse.

For transforming by inverse of an affine transformation (e.g. with scaling) `transform.affine_inverse() * vector` can be used instead. See Transform2D.affine_inverse.

Vector2 **operator ***(right: Vector2) 🔗

Multiplies each component of the **Vector2** by the components of the given **Vector2**.

```
print(Vector2(10, 20) * Vector2(3, 4)) # Prints "(30, 80)"
```

Vector2 **operator ***(right: float) 🔗

Multiplies each component of the **Vector2** by the given float.

Vector2 **operator ***(right: int) 🔗

Multiplies each component of the **Vector2** by the given int.

Vector2 **operator +**(right: Vector2)

Adds each component of the **Vector2** by the components of the given **Vector2**.

```
print(Vector2(10, 20) + Vector2(3, 4)) # Prints "(13, 24)"
```

Vector2 **operator -**(right: Vector2)

Subtracts each component of the **Vector2** by the components of the given **Vector2**.

```
print(Vector2(10, 20) - Vector2(3, 4)) # Prints "(7, 16)"
```

Vector2 **operator /**(right: Vector2)

Divides each component of the **Vector2** by the components of the given **Vector2**.

```
print(Vector2(10, 20) / Vector2(2, 5)) # Prints "(5, 4)"
```

Vector2 **operator /**(right: float)

Divides each component of the **Vector2** by the given float.

Vector2 **operator /**(right: int)

Divides each component of the **Vector2** by the given int.

---

bool **operator <**(right: Vector2)

Compares two **Vector2** vectors by first checking if the X value of the left vector is less than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator <=**(right: Vector2) 🔗

Compares two **Vector2** vectors by first checking if the X value of the left vector is less than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator ==**(right: Vector2) 🔗

Returns `true` if the vectors are exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator >**(right: Vector2) 🔗

Compares two **Vector2** vectors by first checking if the X value of the left vector is greater than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator >=**(right: Vector2) 🔗

Compares two **Vector2** vectors by first checking if the X value of the left vector is greater than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

float **operator []**(index: int) 🔗

Access vector components using their `index`. `v[0]` is equivalent to `v.x`, and `v[1]` is equivalent to `v.y`.

Vector2 **operator unary+**() 🔗

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

Vector2 **operator unary-**() 🔗

Returns the negative value of the **Vector2**. This is the same as writing `Vector2(-v.x, -v.y)`. This operation flips the direction of the vector while keeping the same magnitude. With floats, the number zero can be either positive or negative.

Vector2i

A 2D vector using integer coordinates.

Description

A 2-element structure that can be used to represent 2D grid coordinates or any other pair of integers.

It uses integer coordinates and is therefore preferable to Vector2 when exact precision is required. Note that the values are limited to 32 bits, and unlike Vector2 this cannot be configured with an engine build option. Use int or PackedInt64Array if 64-bit values are needed.

Note: In a boolean context, a Vector2i will evaluate to `false` if it's equal to `Vector2i(0, 0)`. Otherwise, a Vector2i will always evaluate to `true`.

Tutorials

- Math documentation index
- Vector math
- 3Blue1Brown Essence of Linear Algebra [https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab]

Properties

int	x	`0`

int	y	0

Constructors

Vector2i	Vector2i()
Vector2i	Vector2i(from: Vector2i)
Vector2i	Vector2i(from: Vector2)
Vector2i	Vector2i(x: int, y: int)

Methods

Vector2i	abs() const
float	aspect() const
Vector2i	clamp(min: Vector2i, max: Vector2i) const
Vector2i	clampi(min: int, max: int) const
int	distance_squared_to(to: Vector2i) const

float	distance_to(to: Vector2i) const
float	length() const
int	length_squared() const
Vector2i	max(with: Vector2i) const
int	max_axis_index() const
Vector2i	maxi(with: int) const
Vector2i	min(with: Vector2i) const
int	min_axis_index() const
Vector2i	mini(with: int) const
Vector2i	sign() const
Vector2i	snapped(step: Vector2i) const
Vector2i	snappedi(step: int) const

Operators

bool	operator !=(right: Vector2i)
Vector2i	operator %(right: Vector2i)
Vector2i	operator %(right: int)
Vector2i	operator *(right: Vector2i)
Vector2	operator *(right: float)
Vector2i	operator *(right: int)
Vector2i	operator +(right: Vector2i)
Vector2i	operator -(right: Vector2i)
Vector2i	operator /(right: Vector2i)
Vector2	operator /(right: float)
Vector2i	operator /(right: int)
bool	operator <(right: Vector2i)

bool	operator <=(right: Vector2i)
bool	operator ==(right: Vector2i)
bool	operator >(right: Vector2i)
bool	operator >=(right: Vector2i)
int	operator [](index: int)
Vector2i	operator unary+()
Vector2i	operator unary-()

Constants

AXIS_X = `0` 🔗

Enumerated value for the X axis. Returned by max_axis_index and min_axis_index.

AXIS_Y = `1` 🔗

Enumerated value for the Y axis. Returned by max_axis_index and min_axis_index.

ZERO = `Vector2i(0, 0)` 🔗

Zero vector, a vector with all components set to `0`.

ONE = `Vector2i(1, 1)`

One vector, a vector with all components set to `1`.

MIN = `Vector2i(-2147483648, -2147483648)`

Min vector, a vector with all components equal to `INT32_MIN`. Can be used as a negative integer equivalent of Vector2.INF.

MAX = `Vector2i(2147483647, 2147483647)`

Max vector, a vector with all components equal to `INT32_MAX`. Can be used as an integer equivalent of Vector2.INF.

LEFT = `Vector2i(-1, 0)`

Left unit vector. Represents the direction of left.

RIGHT = `Vector2i(1, 0)`

Right unit vector. Represents the direction of right.

UP = `Vector2i(0, -1)`

Up unit vector. Y is down in 2D, so this vector points -Y.

DOWN = `Vector2i(0, 1)`

Down unit vector. Y is down in 2D, so this vector points +Y.

Property Descriptions

int **x** = `0`

The vector's X component. Also accessible by using the index position `[0]`.

---

int **y** = `0` 🔗

The vector's Y component. Also accessible by using the index position `[1]`.

Constructor Descriptions

Vector2i **Vector2i**() 🔗

Constructs a default-initialized **Vector2i** with all components set to `0`.

Vector2i **Vector2i**(from: Vector2i)

Constructs a **Vector2i** as a copy of the given **Vector2i**.

Vector2i **Vector2i**(from: Vector2)

Constructs a new **Vector2i** from the given Vector2 by truncating components' fractional parts (rounding towards zero). For a different behavior consider passing the result of Vector2.ceil, Vector2.floor or Vector2.round to this constructor instead.

Vector2i **Vector2i**(x: int, y: int)

Constructs a new **Vector2i** from the given `x` and `y`.

Method Descriptions

Vector2i **abs**() const 🔗

Returns a new vector with all components in absolute values (i.e. positive).

---

float **aspect**() const 🔗

Returns the aspect ratio of this vector, the ratio of x to y.

Vector2i **clamp**(min: Vector2i, max: Vector2i) const 🔗

Returns a new vector with all components clamped between the components of `min` and `max`, by running @GlobalScope.clamp on each component.

Vector2i **clampi**(min: int, max: int) const 🔗

Returns a new vector with all components clamped between `min` and `max`, by running @GlobalScope.clamp on each component.

---

int **distance_squared_to**(to: Vector2i) const 🔗

Returns the squared distance between this vector and `to`.

This method runs faster than distance_to, so prefer it if you need to compare vectors or need the squared distance for some formula.

---

float **distance_to**(to: Vector2i) const 🔗

Returns the distance between this vector and `to`.

---

float **length**() const

Returns the length (magnitude) of this vector.

---

int **length_squared**() const

Returns the squared length (squared magnitude) of this vector.

This method runs faster than length, so prefer it if you need to compare vectors or need the squared distance for some formula.

Vector2i **max**(with: Vector2i) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector2i(maxi(x, with.x), maxi(y, with.y))`.

---

int **max_axis_index**() const

Returns the axis of the vector's highest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_X.

Vector2i **maxi**(with: int) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector2i(maxi(x, with), maxi(y, with))`.

Vector2i **min**(with: Vector2i) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector2i(mini(x, with.x), mini(y, with.y))`.

---

int **min_axis_index**() const

Returns the axis of the vector's lowest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_Y.

Vector2i **mini**(with: int) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector2i(mini(x, with), mini(y, with))`.

Vector2i **sign**() const

Returns a new vector with each component set to `1` if it's positive, `-1` if it's negative, and `0` if it's zero. The result is identical to calling @GlobalScope.sign on each component.

Vector2i **snapped**(step: Vector2i) const

Returns a new vector with each component snapped to the closest multiple of the corresponding component in `step`.

Vector2i **snappedi**(step: int) const

Returns a new vector with each component snapped to the closest multiple of `step`.

Operator Descriptions

bool **operator !=**(right: Vector2i) 🔗

Returns `true` if the vectors are not equal.

Vector2i **operator %**(right: Vector2i) 🔗

Gets the remainder of each component of the **Vector2i** with the components of the given **Vector2i**. This operation uses truncated division, which is often not desired as it does not work well with negative numbers. Consider using @GlobalScope.posmod instead if you want to handle negative numbers.

```
print(Vector2i(10, -20) % Vector2i(7, 8)) # Prints "(3, -4)"
```

Vector2i **operator %**(right: int) 🔗

Gets the remainder of each component of the **Vector2i** with the given int. This operation uses truncated division, which is often not desired as it does not work well with negative numbers. Consider using @GlobalScope.posmod instead if you want to handle negative numbers.

```
print(Vector2i(10, -20) % 7) # Prints "(3, -6)"
```

Vector2i **operator ***(right: Vector2i) 🔗

Multiplies each component of the **Vector2i** by the components of the given **Vector2i**.

```
print(Vector2i(10, 20) * Vector2i(3, 4)) # Prints "(30, 80)"
```

Vector2 **operator ***(right: float) 🔗

Multiplies each component of the **Vector2i** by the given float. Returns a Vector2.

```
print(Vector2i(10, 15) * 0.9) # Prints "(9, 13.5)"
```

Vector2i **operator ***(right: int)

Multiplies each component of the **Vector2i** by the given int.

Vector2i **operator +**(right: Vector2i)

Adds each component of the **Vector2i** by the components of the given **Vector2i**.

```
print(Vector2i(10, 20) + Vector2i(3, 4)) # Prints "(13, 24)"
```

Vector2i **operator -**(right: Vector2i)

Subtracts each component of the **Vector2i** by the components of the given **Vector2i**.

```
print(Vector2i(10, 20) - Vector2i(3, 4)) # Prints "(7, 16)"
```

Vector2i **operator /**(right: Vector2i)

Divides each component of the **Vector2i** by the components of the given **Vector2i**.

```
print(Vector2i(10, 20) / Vector2i(2, 5)) # Prints "(5, 4)"
```

Vector2 **operator /**(right: float)

Divides each component of the **Vector2i** by the given float. Returns a Vector2.

```
print(Vector2i(10, 20) / 2.9) # Prints "(5, 10)"
```

Vector2i **operator /**(right: int)

Divides each component of the **Vector2i** by the given int.

bool **operator <**(right: Vector2i)

Compares two **Vector2i** vectors by first checking if the X value of the left vector is less than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors. This operator is useful for sorting vectors.

bool **operator <=**(right: Vector2i)

Compares two **Vector2i** vectors by first checking if the X value of the left vector is less than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors. This operator is useful for sorting vectors.

bool **operator ==**(right: Vector2i)

Returns `true` if the vectors are equal.

bool **operator >**(right: Vector2i)

Compares two **Vector2i** vectors by first checking if the X value of the left vector is greater than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors. This operator is useful for sorting vectors.

bool **operator >=**(right: Vector2i)

Compares two **Vector2i** vectors by first checking if the X value of the left vector is greater than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors. This operator is useful for sorting vectors.

---

int **operator []**(index: int)

Access vector components using their `index`. `v[0]` is equivalent to `v.x`, and `v[1]` is equivalent to `v.y`.

Vector2i **operator unary+**()

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

Vector2i **operator unary-**()

Returns the negative value of the **Vector2i**. This is the same as writing `Vector2i(-v.x, -v.y)`. This operation flips the direction of the vector while keeping the same magnitude.

Vector3

A 3D vector using floating-point coordinates.

Description

A 3-element structure that can be used to represent 3D coordinates or any other triplet of numeric values.

It uses floating-point coordinates. By default, these floating-point values use 32-bit precision, unlike float which is always 64-bit. If double precision is needed, compile the engine with the option `precision=double`.

See Vector3i for its integer counterpart.

Note: In a boolean context, a Vector3 will evaluate to `false` if it's equal to `Vector3(0, 0, 0)`. Otherwise, a Vector3 will always evaluate to `true`.

Tutorials

- Math documentation index
- Vector math
- Advanced vector math
- 3Blue1Brown Essence of Linear Algebra [https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab]
- Matrix Transform Demo [https://godotengine.org/asset-library/asset/2787]
- All 3D Demos [https://github.com/godotengine/godot-demo-projects/tree/master/3d]

Properties

float	x	`0.0`
float	y	`0.0`
float	z	`0.0`

Constructors

Vector3	Vector3()
Vector3	Vector3(from: Vector3)
Vector3	Vector3(from: Vector3i)
Vector3	Vector3(x: float, y: float, z: float)

Methods

Vector3	abs() const
float	angle_to(to: Vector3) const

Vector3	bezier_derivative(control_1: Vector3, control_2: Vector3, end: Vector3, t: float) const
Vector3	bezier_interpolate(control_1: Vector3, control_2: Vector3, end: Vector3, t: float) const
Vector3	bounce(n: Vector3) const
Vector3	ceil() const
Vector3	clamp(min: Vector3, max: Vector3) const
Vector3	clampf(min: float, max: float) const
Vector3	cross(with: Vector3) const
Vector3	cubic_interpolate(b: Vector3, pre_a: Vector3, post_b: Vector3, weight: float) const
Vector3	cubic_interpolate_in_time(b: Vector3, pre_a: Vector3, post_b: Vector3, weight: float, b_t: float, pre_a_t: float, post_b_t: float) const
Vector3	direction_to(to: Vector3) const
float	distance_squared_to(to: Vector3) const

float	distance_to(to: Vector3) const
float	dot(with: Vector3) const
Vector3	floor() const
Vector3	inverse() const
bool	is_equal_approx(to: Vector3) const
bool	is_finite() const
bool	is_normalized() const
bool	is_zero_approx() const
float	length() const
float	length_squared() const
Vector3	lerp(to: Vector3, weight: float) const
Vector3	limit_length(length: float = 1.0) const
Vector3	max(with: Vector3) const

int	max_axis_index() const
Vector3	maxf(with: float) const
Vector3	min(with: Vector3) const
int	min_axis_index() const
Vector3	minf(with: float) const
Vector3	move_toward(to: Vector3, delta: float) const
Vector3	normalized() const
Vector3	octahedron_decode(uv: Vector2) static
Vector2	octahedron_encode() const
Basis	outer(with: Vector3) const
Vector3	posmod(mod: float) const
Vector3	posmodv(modv: Vector3) const
Vector3	project(b: Vector3) const

Vector3	reflect(n: Vector3) const
Vector3	rotated(axis: Vector3, angle: float) const
Vector3	round() const
Vector3	sign() const
float	signed_angle_to(to: Vector3, axis: Vector3) const
Vector3	slerp(to: Vector3, weight: float) const
Vector3	slide(n: Vector3) const
Vector3	snapped(step: Vector3) const
Vector3	snappedf(step: float) const

Operators

bool	operator !=(right: Vector3)
Vector3	operator *(right: Basis)

Vector3	operator *(right: Quaternion)
Vector3	operator *(right: Transform3D)
Vector3	operator *(right: Vector3)
Vector3	operator *(right: float)
Vector3	operator *(right: int)
Vector3	operator +(right: Vector3)
Vector3	operator -(right: Vector3)
Vector3	operator /(right: Vector3)
Vector3	operator /(right: float)
Vector3	operator /(right: int)
bool	operator <(right: Vector3)
bool	operator <=(right: Vector3)
bool	operator ==(right: Vector3)

bool	operator >(right: Vector3)
bool	operator >=(right: Vector3)
float	operator [](index: int)
Vector3	operator unary+()
Vector3	operator unary-()

Constants

AXIS_X = `0`

Enumerated value for the X axis. Returned by max_axis_index and min_axis_index.

AXIS_Y = `1`

Enumerated value for the Y axis. Returned by max_axis_index and min_axis_index.

AXIS_Z = `2`

Enumerated value for the Z axis. Returned by max_axis_index and min_axis_index.

ZERO = `Vector3(0, 0, 0)`

Zero vector, a vector with all components set to `0`.

ONE = `Vector3(1, 1, 1)`

One vector, a vector with all components set to `1`.

INF = `Vector3(inf, inf, inf)`

Infinity vector, a vector with all components set to @GDScript.INF.

LEFT = `Vector3(-1, 0, 0)`

Left unit vector. Represents the local direction of left, and the global direction of west.

RIGHT = `Vector3(1, 0, 0)`

Right unit vector. Represents the local direction of right, and the global direction of east.

UP = `Vector3(0, 1, 0)`

Up unit vector.

DOWN = `Vector3(0, -1, 0)`

Down unit vector.

FORWARD = `Vector3(0, 0, -1)`

Forward unit vector. Represents the local direction of forward, and the global direction of north. Keep in mind that the forward direction for lights, cameras, etc is different from 3D assets like characters, which face towards the camera by convention. Use MODEL_FRONT and similar constants when working in 3D asset space.

BACK = `Vector3(0, 0, 1)`

Back unit vector. Represents the local direction of back, and the global direction of south.

MODEL_LEFT = `Vector3(1, 0, 0)`

Unit vector pointing towards the left side of imported 3D assets.

MODEL_RIGHT = `Vector3(-1, 0, 0)`

Unit vector pointing towards the right side of imported 3D assets.

MODEL_TOP = `Vector3(0, 1, 0)`

Unit vector pointing towards the top side (up) of imported 3D assets.

MODEL_BOTTOM = `Vector3(0, -1, 0)`

Unit vector pointing towards the bottom side (down) of imported 3D assets.

MODEL_FRONT = `Vector3(0, 0, 1)`

Unit vector pointing towards the front side (facing forward) of imported 3D assets.

MODEL_REAR = `Vector3(0, 0, -1)`

Unit vector pointing towards the rear side (back) of imported 3D assets.

Property Descriptions

float **x** = `0.0`

The vector's X component. Also accessible by using the index position `[0]`.

float **y** = `0.0`

The vector's Y component. Also accessible by using the index position `[1]`.

float **z** = `0.0`

The vector's Z component. Also accessible by using the index position `[2]`.

Constructor Descriptions

Vector3 **Vector3**()

Constructs a default-initialized **Vector3** with all components set to `0`.

Vector3 **Vector3**(from: Vector3)

Constructs a **Vector3** as a copy of the given **Vector3**.

Vector3 **Vector3**(from: Vector3i)

Constructs a new **Vector3** from Vector3i.

Vector3 **Vector3**(x: float, y: float, z: float)

Returns a **Vector3** with the given components.

Method Descriptions

Vector3 **abs**() const

Returns a new vector with all components in absolute values (i.e. positive).

float **angle_to**(to: Vector3) const

Returns the unsigned minimum angle to the given vector, in radians.

Vector3 **bezier_derivative**(control_1: Vector3, control_2: Vector3, end: Vector3, t: float) const

Returns the derivative at the given `t` on the Bézier curve [https://en.wikipedia.org/wiki/B%C3%A9zier_curve] defined by this vector and the given `control_1`, `control_2`, and `end` points.

Vector3 **bezier_interpolate**(control_1: Vector3, control_2: Vector3, end: Vector3, t: float) const

Returns the point at the given `t` on the Bézier curve [https://en.wikipedia.org/wiki/B%C3%A9zier_curve] defined by this vector and the given `control_1`, `control_2`, and `end` points.

Vector3 **bounce**(n: Vector3) const

Returns the vector "bounced off" from a plane defined by the given normal `n`.

Note: bounce performs the operation that most engines and frameworks call `reflect()`.

Vector3 **ceil**() const 🔗

Returns a new vector with all components rounded up (towards positive infinity).

Vector3 **clamp**(min: Vector3, max: Vector3) const 🔗

Returns a new vector with all components clamped between the components of `min` and `max`, by running @GlobalScope.clamp on each component.

Vector3 **clampf**(min: float, max: float) const 🔗

Returns a new vector with all components clamped between `min` and `max`, by running @GlobalScope.clamp on each component.

Vector3 **cross**(with: Vector3) const 🔗

Returns the cross product of this vector and `with`.

This returns a vector perpendicular to both this and `with`, which would be the normal vector of the plane defined by the two vectors. As there are two such vectors, in opposite directions, this method returns the vector defined by a right-handed coordinate system. If the two vectors are parallel this returns an empty vector, making it useful for testing if two vectors are parallel.

Vector3 **cubic_interpolate**(b: Vector3, pre_a: Vector3, post_b: Vector3, weight: float) const

Performs a cubic interpolation between this vector and `b` using `pre_a` and `post_b` as handles, and returns the result at position `weight`. `weight` is on the range of 0.0 to 1.0, representing the amount of interpolation.

Vector3 **cubic_interpolate_in_time**(b: Vector3, pre_a: Vector3, post_b: Vector3, weight: float, b_t: float, pre_a_t: float, post_b_t: float) const

Performs a cubic interpolation between this vector and `b` using `pre_a` and `post_b` as handles, and returns the result at position `weight`. `weight` is on the range of 0.0 to 1.0, representing the amount of interpolation.

It can perform smoother interpolation than cubic_interpolate by the time values.

Vector3 **direction_to**(to: Vector3) const

Returns the normalized vector pointing from this vector to `to`. This is equivalent to using `(b - a).normalized()`.

float **distance_squared_to**(to: Vector3) const

Returns the squared distance between this vector and `to`.

This method runs faster than distance_to, so prefer it if you need to compare vectors or need the squared distance for some formula.

float **distance_to**(to: Vector3) const 🔗

Returns the distance between this vector and `to`.

---

float **dot**(with: Vector3) const 🔗

Returns the dot product of this vector and `with`. This can be used to compare the angle between two vectors. For example, this can be used to determine whether an enemy is facing the player.

The dot product will be `0` for a right angle (90 degrees), greater than 0 for angles narrower than 90 degrees and lower than 0 for angles wider than 90 degrees.

When using unit (normalized) vectors, the result will always be between `-1.0` (180 degree angle) when the vectors are facing opposite directions, and `1.0` (0 degree angle) when the vectors are aligned.

Note: `a.dot(b)` is equivalent to `b.dot(a)`.

Vector3 **floor**() const 🔗

Returns a new vector with all components rounded down (towards negative infinity).

Vector3 **inverse**() const 🔗

Returns the inverse of the vector. This is the same as `Vector3(1.0 / v.x, 1.0 / v.y, 1.0 / v.z)`.

---

bool **is_equal_approx**(to: Vector3) const 🔗

Returns `true` if this vector and `to` are approximately equal, by running @GlobalScope.is_equal_approx on each component.

---

bool **is_finite**() const 🔗

Returns `true` if this vector is finite, by calling @GlobalScope.is_finite on each component.

---

bool **is_normalized**() const 🔗

Returns `true` if the vector is normalized, i.e. its length is approximately equal to 1.

---

bool **is_zero_approx**() const 🔗

Returns `true` if this vector's values are approximately zero, by running @GlobalScope.is_zero_approx on each component.

This method is faster than using is_equal_approx with one value as a zero vector.

---

float **length**() const 🔗

Returns the length (magnitude) of this vector.

---

float **length_squared**() const 🔗

Returns the squared length (squared magnitude) of this vector.

This method runs faster than length, so prefer it if you need to compare vectors or need the squared distance for some formula.

Vector3 **lerp**(to: Vector3, weight: float) const

Returns the result of the linear interpolation between this vector and `to` by amount `weight`. `weight` is on the range of `0.0` to `1.0`, representing the amount of interpolation.

Vector3 **limit_length**(length: float = 1.0) const

Returns the vector with a maximum length by limiting its length to `length`.

Vector3 **max**(with: Vector3) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector3(maxf(x, with.x), maxf(y, with.y), maxf(z, with.z))`.

int **max_axis_index**() const

Returns the axis of the vector's highest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_X.

Vector3 **maxf**(with: float) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector3(maxf(x, with), maxf(y, with), maxf(z, with))`.

Vector3 **min**(with: Vector3) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector3(minf(x, with.x), minf(y, with.y), minf(z, with.z))`.

int **min_axis_index**() const

Returns the axis of the vector's lowest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_Z.

Vector3 **minf**(with: float) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector3(minf(x, with), minf(y, with), minf(z, with))`.

Vector3 **move_toward**(to: Vector3, delta: float) const

Returns a new vector moved toward `to` by the fixed `delta` amount. Will not go past the final value.

Vector3 **normalized**() const

Returns the result of scaling the vector to unit length. Equivalent to `v / v.length()`. Returns `(0, 0, 0)` if `v.length() == 0`. See also is_normalized.

Note: This function may return incorrect values if the input vector length is near zero.

Vector3 **octahedron_decode**(uv: Vector2) static

Returns the **Vector3** from an octahedral-compressed form created using octahedron_encode (stored as a Vector2).

Vector2 **octahedron_encode**() const

Returns the octahedral-encoded (oct32) form of this **Vector3** as a Vector2. Since a Vector2 occupies 1/3 less memory compared to **Vector3**, this form of compression can be used to pass greater amounts of normalized **Vector3**s without increasing storage or memory requirements. See also octahedron_decode.

Note: octahedron_encode can only be used for normalized vectors. octahedron_encode does *not* check whether this **Vector3** is normalized, and will return a value that does not decompress to the original value if the **Vector3** is not normalized.

Note: Octahedral compression is *lossy*, although visual differences are rarely perceptible in real world scenarios.

Basis **outer**(with: Vector3) const 🔗

Returns the outer product with `with`.

Vector3 **posmod**(mod: float) const 🔗

Returns a vector composed of the @GlobalScope.fposmod of this vector's components and `mod`.

Vector3 **posmodv**(modv: Vector3) const 🔗

Returns a vector composed of the @GlobalScope.fposmod of this vector's components and `modv`'s components.

Vector3 **project**(b: Vector3) const 🔗

Returns a new vector resulting from projecting this vector onto the given vector `b`. The resulting new vector is parallel to `b`. See also slide.

Note: If the vector `b` is a zero vector, the components of the resulting new vector will be @GDScript.NAN.

Vector3 **reflect**(n: Vector3) const

Returns the result of reflecting the vector through a plane defined by the given normal vector `n`.

Note: reflect differs from what other engines and frameworks call `reflect()`. In other engines, `reflect()` returns the result of the vector reflected by the given plane. The reflection thus passes through the given normal. While in Godot the reflection passes through the plane and can be thought of as bouncing off the normal. See also bounce which does what most engines call `reflect()`.

Vector3 **rotated**(axis: Vector3, angle: float) const

Returns the result of rotating this vector around a given axis by `angle` (in radians). The axis must be a normalized vector. See also @GlobalScope.deg_to_rad.

Vector3 **round**() const

Returns a new vector with all components rounded to the nearest integer, with halfway cases rounded away from zero.

Vector3 **sign**() const

Returns a new vector with each component set to `1.0` if it's positive, `-1.0` if it's negative, and `0.0` if it's zero. The result is identical to calling @GlobalScope.sign on each component.

---

float **signed_angle_to**(to: Vector3, axis: Vector3) const

Returns the signed angle to the given vector, in radians. The sign of the angle is positive in a counter-clockwise direction and negative in a clockwise direction when viewed from the side specified by the `axis`.

Vector3 **slerp**(to: Vector3, weight: float) const

Returns the result of spherical linear interpolation between this vector and `to`, by amount `weight`. `weight` is on the range of 0.0 to 1.0, representing the amount of interpolation.

This method also handles interpolating the lengths if the input vectors have different lengths. For the special case of one or both input vectors having zero length, this method behaves like lerp.

Vector3 **slide**(n: Vector3) const

Returns a new vector resulting from sliding this vector along a plane with normal `n`. The resulting new vector is perpendicular to `n`, and is equivalent to this vector minus its projection on `n`. See also project.

Note: The vector `n` must be normalized. See also normalized.

Vector3 **snapped**(step: Vector3) const

Returns a new vector with each component snapped to the nearest multiple of the corresponding component in `step`. This can also be used to round the components to an arbitrary number of decimals.

Vector3 **snappedf**(step: float) const

Returns a new vector with each component snapped to the nearest multiple of `step`. This can also be used to round the components to an arbitrary number of decimals.

Operator Descriptions

bool **operator !=**(right: Vector3) 🔗

Returns `true` if the vectors are not equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

Vector3 **operator ***(right: Basis) 🔗

Inversely transforms (multiplies) the **Vector3** by the given Basis matrix, under the assumption that the basis is orthonormal (i.e. rotation/reflection is fine, scaling/skew is not).

`vector * basis` is equivalent to `basis.transposed() * vector`. See Basis.transposed.

For transforming by inverse of a non-orthonormal basis (e.g. with scaling) `basis.inverse() * vector` can be used instead. See Basis.inverse.

Vector3 **operator ***(right: Quaternion) 🔗

Inversely transforms (multiplies) the **Vector3** by the given Quaternion.

`vector * quaternion` is equivalent to `quaternion.inverse() * vector`. See Quaternion.inverse.

Vector3 **operator ***(right: Transform3D)

Inversely transforms (multiplies) the **Vector3** by the given Transform3D transformation matrix, under the assumption that the transformation basis is orthonormal (i.e. rotation/reflection is fine, scaling/skew is not).

`vector * transform` is equivalent to `transform.inverse() * vector`. See Transform3D.inverse.

For transforming by inverse of an affine transformation (e.g. with scaling) `transform.affine_inverse() * vector` can be used instead. See Transform3D.affine_inverse.

Vector3 **operator ***(right: Vector3)

Multiplies each component of the **Vector3** by the components of the given **Vector3**.

```
print(Vector3(10, 20, 30) * Vector3(3, 4, 5)) # Prints "(30, 80, 150)"
```

Vector3 **operator ***(right: float)

Multiplies each component of the **Vector3** by the given float.

Vector3 **operator ***(right: int)

Multiplies each component of the **Vector3** by the given int.

Vector3 **operator +**(right: Vector3)

Adds each component of the **Vector3** by the components of the given **Vector3**.

```
print(Vector3(10, 20, 30) + Vector3(3, 4, 5)) # Prints "(13, 24, 35)"
```

Vector3 **operator -**(right: Vector3)

Subtracts each component of the **Vector3** by the components of the given **Vector3**.

```
print(Vector3(10, 20, 30) - Vector3(3, 4, 5)) # Prints "(7, 16, 25)"
```

Vector3 **operator /**(right: Vector3)

Divides each component of the **Vector3** by the components of the given **Vector3**.

```
print(Vector3(10, 20, 30) / Vector3(2, 5, 3)) # Prints "(5, 4, 10)"
```

Vector3 **operator /**(right: float)

Divides each component of the **Vector3** by the given float.

Vector3 **operator /**(right: int)

Divides each component of the **Vector3** by the given int.

---

bool **operator <**(right: Vector3)

Compares two **Vector3** vectors by first checking if the X value of the left vector is less than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, and then with the Z values. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator <=**(right: Vector3)

Compares two **Vector3** vectors by first checking if the X value of the left vector is less than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, and then with the Z values. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator ==**(right: Vector3)

Returns `true` if the vectors are exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator >**(right: Vector3)

Compares two **Vector3** vectors by first checking if the X value of the left vector is greater than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, and then with the Z values. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator >=**(right: Vector3)

Compares two **Vector3** vectors by first checking if the X value of the left vector is greater than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, and then with the Z values. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

float **operator []**(index: int)

Access vector components using their `index`. `v[0]` is equivalent to `v.x`, `v[1]` is equivalent to `v.y`, and `v[2]` is equivalent to `v.z`.

Vector3 **operator unary+**()

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

Vector3 **operator unary-()** 🔗

Returns the negative value of the **Vector3**. This is the same as writing `Vector3(-v.x, -v.y, -v.z)`. This operation flips the direction of the vector while keeping the same magnitude. With floats, the number zero can be either positive or negative.

Vector3i

A 3D vector using integer coordinates.

Description

A 3-element structure that can be used to represent 3D grid coordinates or any other triplet of integers.

It uses integer coordinates and is therefore preferable to Vector3 when exact precision is required. Note that the values are limited to 32 bits, and unlike Vector3 this cannot be configured with an engine build option. Use int or PackedInt64Array if 64-bit values are needed.

Note: In a boolean context, a Vector3i will evaluate to `false` if it's equal to `Vector3i(0, 0, 0)`. Otherwise, a Vector3i will always evaluate to `true`.

Tutorials

- Math documentation index
- Vector math
- 3Blue1Brown Essence of Linear Algebra [https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab]

Properties

int x `0`

int	y	0
int	z	0

Constructors

Vector3i	Vector3i()
Vector3i	Vector3i(from: Vector3i)
Vector3i	Vector3i(from: Vector3)
Vector3i	Vector3i(x: int, y: int, z: int)

Methods

Vector3i	abs() const
Vector3i	clamp(min: Vector3i, max: Vector3i) const
Vector3i	clampi(min: int, max: int) const
int	distance_squared_to(to: Vector3i) const

float	distance_to(to: Vector3i) const
float	length() const
int	length_squared() const
Vector3i	max(with: Vector3i) const
int	max_axis_index() const
Vector3i	maxi(with: int) const
Vector3i	min(with: Vector3i) const
int	min_axis_index() const
Vector3i	mini(with: int) const
Vector3i	sign() const
Vector3i	snapped(step: Vector3i) const
Vector3i	snappedi(step: int) const

Operators

bool	operator !=(right: Vector3i)
Vector3i	operator %(right: Vector3i)
Vector3i	operator %(right: int)
Vector3i	operator *(right: Vector3i)
Vector3	operator *(right: float)
Vector3i	operator *(right: int)
Vector3i	operator +(right: Vector3i)
Vector3i	operator -(right: Vector3i)
Vector3i	operator /(right: Vector3i)
Vector3	operator /(right: float)
Vector3i	operator /(right: int)
bool	operator <(right: Vector3i)

bool	operator <=(right: Vector3i)
bool	operator ==(right: Vector3i)
bool	operator >(right: Vector3i)
bool	operator >=(right: Vector3i)
int	operator [](index: int)
Vector3i	operator unary+()
Vector3i	operator unary-()

Constants

AXIS_X = `0` 🔗

Enumerated value for the X axis. Returned by max_axis_index and min_axis_index.

AXIS_Y = `1` 🔗

Enumerated value for the Y axis. Returned by max_axis_index and min_axis_index.

AXIS_Z = `2` 🔗

Enumerated value for the Z axis. Returned by max_axis_index and min_axis_index.

ZERO = `Vector3i(0, 0, 0)`

Zero vector, a vector with all components set to `0`.

ONE = `Vector3i(1, 1, 1)`

One vector, a vector with all components set to `1`.

MIN = `Vector3i(-2147483648, -2147483648, -2147483648)`

Min vector, a vector with all components equal to `INT32_MIN`. Can be used as a negative integer equivalent of Vector3.INF.

MAX = `Vector3i(2147483647, 2147483647, 2147483647)`

Max vector, a vector with all components equal to `INT32_MAX`. Can be used as an integer equivalent of Vector3.INF.

LEFT = `Vector3i(-1, 0, 0)`

Left unit vector. Represents the local direction of left, and the global direction of west.

RIGHT = `Vector3i(1, 0, 0)`

Right unit vector. Represents the local direction of right, and the global direction of east.

UP = `Vector3i(0, 1, 0)`

Up unit vector.

DOWN = `Vector3i(0, -1, 0)`

Down unit vector.

FORWARD = `Vector3i(0, 0, -1)`

Forward unit vector. Represents the local direction of forward, and the global direction of north.

BACK = `Vector3i(0, 0, 1)`

Back unit vector. Represents the local direction of back, and the global direction of south.

Property Descriptions

int **x** = `0`

The vector's X component. Also accessible by using the index position `[0]`.

---

int **y** = `0`

The vector's Y component. Also accessible by using the index position `[1]`.

---

int **z** = `0`

The vector's Z component. Also accessible by using the index position `[2]`.

Constructor Descriptions

Vector3i **Vector3i**()

Constructs a default-initialized **Vector3i** with all components set to `0`.

Vector3i **Vector3i**(from: Vector3i)

Constructs a **Vector3i** as a copy of the given **Vector3i**.

Vector3i **Vector3i**(from: Vector3)

Constructs a new **Vector3i** from the given Vector3 by truncating components' fractional parts (rounding towards zero). For a different behavior consider passing the result of Vector3.ceil, Vector3.floor or Vector3.round to this constructor instead.

Vector3i **Vector3i**(x: int, y: int, z: int)

Returns a **Vector3i** with the given components.

Method Descriptions

Vector3i **abs**() const

Returns a new vector with all components in absolute values (i.e. positive).

Vector3i **clamp**(min: Vector3i, max: Vector3i) const

Returns a new vector with all components clamped between the components of `min` and `max`, by running @GlobalScope.clamp on each component.

Vector3i **clampi**(min: int, max: int) const

Returns a new vector with all components clamped between `min` and `max`, by running @GlobalScope.clamp on each component.

int **distance_squared_to**(to: Vector3i) const

Returns the squared distance between this vector and `to`.

This method runs faster than distance_to, so prefer it if you need to compare vectors or need the squared distance for some formula.

float **distance_to**(to: Vector3i) const

Returns the distance between this vector and `to`.

float **length**() const

Returns the length (magnitude) of this vector.

int **length_squared**() const

Returns the squared length (squared magnitude) of this vector.

This method runs faster than length, so prefer it if you need to compare vectors or need the squared distance for some formula.

Vector3i **max**(with: Vector3i) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector3i(maxi(x, with.x), maxi(y, with.y), maxi(z, with.z))`.

int **max_axis_index**() const

Returns the axis of the vector's highest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_X.

Vector3i **maxi**(with: int) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector3i(maxi(x, with), maxi(y, with), maxi(z, with))`.

Vector3i **min**(with: Vector3i) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector3i(mini(x, with.x), mini(y, with.y), mini(z, with.z))`.

int **min_axis_index**() const

Returns the axis of the vector's lowest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_Z.

Vector3i **mini**(with: int) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector3i(mini(x, with), mini(y, with), mini(z, with))`.

Vector3i **sign**() const

Returns a new vector with each component set to `1` if it's positive, `-1` if it's negative, and `0` if it's zero. The result is identical to calling @GlobalScope.sign on each component.

Vector3i **snapped**(step: Vector3i) const

Returns a new vector with each component snapped to the closest multiple of the corresponding component in `step`.

Vector3i **snappedi**(step: int) const

Returns a new vector with each component snapped to the closest multiple of `step`.

Operator Descriptions

bool **operator !=**(right: Vector3i)

Returns `true` if the vectors are not equal.

Vector3i **operator %**(right: Vector3i)

Gets the remainder of each component of the **Vector3i** with the components of the given **Vector3i**. This operation uses truncated division, which is often not desired as it does not work well with negative numbers. Consider using @GlobalScope.posmod instead if you want to handle negative numbers.

```
print(Vector3i(10, -20, 30) % Vector3i(7, 8, 9)) # Prints "(3, -4, 3)"
```

Vector3i **operator %**(right: int) 🔗

Gets the remainder of each component of the **Vector3i** with the given int. This operation uses truncated division, which is often not desired as it does not work well with negative numbers. Consider using @GlobalScope.posmod instead if you want to handle negative numbers.

```
print(Vector3i(10, -20, 30) % 7) # Prints "(3, -6, 2)"
```

Vector3i **operator ***(right: Vector3i) 🔗

Multiplies each component of the **Vector3i** by the components of the given **Vector3i**.

```
print(Vector3i(10, 20, 30) * Vector3i(3, 4, 5)) # Prints "(30, 80, 150)"
```

Vector3 **operator ***(right: float) 🔗

Multiplies each component of the **Vector3i** by the given float. Returns a Vector3.

```
print(Vector3i(10, 15, 20) * 0.9) # Prints "(9, 13.5, 18)"
```

Vector3i **operator ***(right: int) 🔗

Multiplies each component of the **Vector3i** by the given int.

Vector3i **operator +**(right: Vector3i) 🔗

Adds each component of the **Vector3i** by the components of the given **Vector3i**.

```
print(Vector3i(10, 20, 30) + Vector3i(3, 4, 5)) # Prints "(13, 24, 35)"
```

Vector3i **operator -**(right: Vector3i)

Subtracts each component of the **Vector3i** by the components of the given **Vector3i**.

```
print(Vector3i(10, 20, 30) - Vector3i(3, 4, 5)) # Prints "(7, 16, 25)"
```

Vector3i **operator /**(right: Vector3i)

Divides each component of the **Vector3i** by the components of the given **Vector3i**.

```
print(Vector3i(10, 20, 30) / Vector3i(2, 5, 3)) # Prints "(5, 4, 10)"
```

Vector3 **operator /**(right: float)

Divides each component of the **Vector3i** by the given float. Returns a Vector3.

```
print(Vector3i(10, 20, 30) / 2.9) # Prints "(5, 10, 15)"
```

Vector3i **operator /**(right: int)

Divides each component of the **Vector3i** by the given int.

bool **operator <**(right: Vector3i)

Compares two **Vector3i** vectors by first checking if the X value of the left vector is less than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, and then with the Z values. This operator is useful for sorting vectors.

---

bool **operator <=**(right: Vector3i) 🔗

Compares two **Vector3i** vectors by first checking if the X value of the left vector is less than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, and then with the Z values. This operator is useful for sorting vectors.

---

bool **operator ==**(right: Vector3i) 🔗

Returns `true` if the vectors are equal.

---

bool **operator >**(right: Vector3i) 🔗

Compares two **Vector3i** vectors by first checking if the X value of the left vector is greater than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, and then with the Z values. This operator is useful for sorting vectors.

---

bool **operator >=**(right: Vector3i) 🔗

Compares two **Vector3i** vectors by first checking if the X value of the left vector is greater than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, and then with the Z values. This operator is useful for sorting vectors.

---

int **operator []**(index: int)

Access vector components using their `index`. `v[0]` is equivalent to `v.x`, `v[1]` is equivalent to `v.y`, and `v[2]` is equivalent to `v.z`.

Vector3i **operator unary+**()

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

Vector3i **operator unary-**()

Returns the negative value of the **Vector3i**. This is the same as writing `Vector3i(-v.x, -v.y, -v.z)`. This operation flips the direction of the vector while keeping the same magnitude.

Vector4

A 4D vector using floating-point coordinates.

Description

A 4-element structure that can be used to represent 4D coordinates or any other quadruplet of numeric values.

It uses floating-point coordinates. By default, these floating-point values use 32-bit precision, unlike float which is always 64-bit. If double precision is needed, compile the engine with the option `precision=double`.

See Vector4i for its integer counterpart.

Note: In a boolean context, a Vector4 will evaluate to `false` if it's equal to `Vector4(0, 0, 0, 0)`. Otherwise, a Vector4 will always evaluate to `true`.

Properties

float	w	`0.0`
float	x	`0.0`
float	y	`0.0`

float z `0.0`

Constructors

Vector4 Vector4()

Vector4 Vector4(from: Vector4)

Vector4 Vector4(from: Vector4i)

Vector4 Vector4(x: float, y: float, z: float, w: float)

Methods

Vector4 abs() const

Vector4 ceil() const

Vector4 clamp(min: Vector4, max: Vector4) const

Vector4 clampf(min: float, max: float) const

Vector4	cubic_interpolate(b: Vector4, pre_a: Vector4, post_b: Vector4, weight: float) const
Vector4	cubic_interpolate_in_time(b: Vector4, pre_a: Vector4, post_b: Vector4, weight: float, b_t: float, pre_a_t: float, post_b_t: float) const
Vector4	direction_to(to: Vector4) const
float	distance_squared_to(to: Vector4) const
float	distance_to(to: Vector4) const
float	dot(with: Vector4) const
Vector4	floor() const
Vector4	inverse() const
bool	is_equal_approx(to: Vector4) const
bool	is_finite() const
bool	is_normalized() const
bool	is_zero_approx() const

float	length() const
float	length_squared() const
Vector4	lerp(to: Vector4, weight: float) const
Vector4	max(with: Vector4) const
int	max_axis_index() const
Vector4	maxf(with: float) const
Vector4	min(with: Vector4) const
int	min_axis_index() const
Vector4	minf(with: float) const
Vector4	normalized() const
Vector4	posmod(mod: float) const
Vector4	posmodv(modv: Vector4) const
Vector4	round() const

Vector4	sign() const
Vector4	snapped(step: Vector4) const
Vector4	snappedf(step: float) const

Operators

bool	operator !=(right: Vector4)
Vector4	operator *(right: Projection)
Vector4	operator *(right: Vector4)
Vector4	operator *(right: float)
Vector4	operator *(right: int)
Vector4	operator +(right: Vector4)
Vector4	operator -(right: Vector4)
Vector4	operator /(right: Vector4)

Vector4	operator /(right: float)
Vector4	operator /(right: int)
bool	operator <(right: Vector4)
bool	operator <=(right: Vector4)
bool	operator ==(right: Vector4)
bool	operator >(right: Vector4)
bool	operator >=(right: Vector4)
float	operator [](index: int)
Vector4	operator unary+()
Vector4	operator unary-()

Constants

AXIS_X = 0 🔗

Enumerated value for the X axis. Returned by max_axis_index and min_axis_index.

AXIS_Y = `1`

Enumerated value for the Y axis. Returned by max_axis_index and min_axis_index.

AXIS_Z = `2`

Enumerated value for the Z axis. Returned by max_axis_index and min_axis_index.

AXIS_W = `3`

Enumerated value for the W axis. Returned by max_axis_index and min_axis_index.

ZERO = `Vector4(0, 0, 0, 0)`

Zero vector, a vector with all components set to `0`.

ONE = `Vector4(1, 1, 1, 1)`

One vector, a vector with all components set to `1`.

INF = `Vector4(inf, inf, inf, inf)`

Infinity vector, a vector with all components set to @GDScript.INF.

Property Descriptions

float **w** = `0.0`

The vector's W component. Also accessible by using the index position `[3]`.

---

float **x** = `0.0` 🔗

The vector's X component. Also accessible by using the index position `[0]`.

---

float **y** = `0.0` 🔗

The vector's Y component. Also accessible by using the index position `[1]`.

---

float **z** = `0.0` 🔗

The vector's Z component. Also accessible by using the index position `[2]`.

Constructor Descriptions

Vector4 **Vector4**() 🔗

Constructs a default-initialized **Vector4** with all components set to `0`.

Vector4 **Vector4**(from: Vector4)

Constructs a **Vector4** as a copy of the given **Vector4**.

Vector4 **Vector4**(from: Vector4i)

Constructs a new **Vector4** from the given Vector4i.

Vector4 **Vector4**(x: float, y: float, z: float, w: float)

Returns a **Vector4** with the given components.

Method Descriptions

Vector4 **abs**() const 🔗

Returns a new vector with all components in absolute values (i.e. positive).

Vector4 **ceil**() const 🔗

Returns a new vector with all components rounded up (towards positive infinity).

Vector4 **clamp**(min: Vector4, max: Vector4) const 🔗

Returns a new vector with all components clamped between the components of `min` and `max`, by running @GlobalScope.clamp on each component.

Vector4 **clampf**(min: float, max: float) const 🔗

Returns a new vector with all components clamped between `min` and `max`, by running @GlobalScope.clamp on each component.

Vector4 **cubic_interpolate**(b: Vector4, pre_a: Vector4, post_b: Vector4, weight: float) const

Performs a cubic interpolation between this vector and `b` using `pre_a` and `post_b` as handles, and returns the result at position `weight`. `weight` is on the range of 0.0 to 1.0, representing the amount of interpolation.

Vector4 **cubic_interpolate_in_time**(b: Vector4, pre_a: Vector4, post_b: Vector4, weight: float, b_t: float, pre_a_t: float, post_b_t: float) const

Performs a cubic interpolation between this vector and `b` using `pre_a` and `post_b` as handles, and returns the result at position `weight`. `weight` is on the range of 0.0 to 1.0, representing the amount of interpolation.

It can perform smoother interpolation than cubic_interpolate by the time values.

Vector4 **direction_to**(to: Vector4) const

Returns the normalized vector pointing from this vector to `to`. This is equivalent to using `(b - a).normalized()`.

---

float **distance_squared_to**(to: Vector4) const

Returns the squared distance between this vector and `to`.

This method runs faster than distance_to, so prefer it if you need to compare vectors or need the squared distance for some formula.

---

float **distance_to**(to: Vector4) const

Returns the distance between this vector and `to`.

float **dot**(with: Vector4) const

Returns the dot product of this vector and `with`.

Vector4 **floor**() const

Returns a new vector with all components rounded down (towards negative infinity).

Vector4 **inverse**() const

Returns the inverse of the vector. This is the same as `Vector4(1.0 / v.x, 1.0 / v.y, 1.0 / v.z, 1.0 / v.w)`.

bool **is_equal_approx**(to: Vector4) const

Returns `true` if this vector and `to` are approximately equal, by running @GlobalScope.is_equal_approx on each component.

bool **is_finite**() const

Returns `true` if this vector is finite, by calling @GlobalScope.is_finite on each component.

bool **is_normalized**() const

Returns `true` if the vector is normalized, i.e. its length is approximately equal to 1.

bool **is_zero_approx**() const

Returns `true` if this vector's values are approximately zero, by running @GlobalScope.is_zero_approx on each component.

This method is faster than using is_equal_approx with one value as a zero vector.

float **length**() const

Returns the length (magnitude) of this vector.

float **length_squared**() const

Returns the squared length (squared magnitude) of this vector.

This method runs faster than length, so prefer it if you need to compare vectors or need the squared distance for some formula.

Vector4 **lerp**(to: Vector4, weight: float) const

Returns the result of the linear interpolation between this vector and `to` by amount `weight`. `weight` is on the range of `0.0` to `1.0`, representing the amount of interpolation.

Vector4 **max**(with: Vector4) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector4(maxf(x, with.x), maxf(y, with.y), maxf(z, with.z),` `maxf(w, with.w))`.

---

int **max_axis_index**() const

Returns the axis of the vector's highest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_X.

Vector4 **maxf**(with: float) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector4(maxf(x, with), maxf(y, with), maxf(z, with), maxf(w, with))`.

Vector4 **min**(with: Vector4) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector4(minf(x, with.x), minf(y, with.y), minf(z, with.z), minf(w, with.w))`.

---

int **min_axis_index**() const

Returns the axis of the vector's lowest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_W.

Vector4 **minf**(with: float) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector4(minf(x, with), minf(y, with), minf(z, with), minf(w, with))`.

Vector4 **normalized**() const 🔗

Returns the result of scaling the vector to unit length. Equivalent to `v / v.length()`. Returns `(0, 0, 0, 0)` if `v.length() == 0`. See also is_normalized.

Note: This function may return incorrect values if the input vector length is near zero.

Vector4 **posmod**(mod: float) const 🔗

Returns a vector composed of the @GlobalScope.fposmod of this vector's components and `mod`.

Vector4 **posmodv**(modv: Vector4) const 🔗

Returns a vector composed of the @GlobalScope.fposmod of this vector's components and `modv`'s components.

Vector4 **round**() const 🔗

Returns a new vector with all components rounded to the nearest integer, with halfway cases rounded away from zero.

Vector4 **sign**() const 🔗

Returns a new vector with each component set to `1.0` if it's positive, `-1.0` if it's negative, and `0.0` if it's zero. The result is identical to calling @GlobalScope.sign on each component.

Vector4 **snapped**(step: Vector4) const 🔗

Returns a new vector with each component snapped to the nearest multiple of the corresponding component in `step`. This can also be used to round the components to an arbitrary number of decimals.

Vector4 **snappedf**(step: float) const

Returns a new vector with each component snapped to the nearest multiple of `step`. This can also be used to round the components to an arbitrary number of decimals.

Operator Descriptions

bool **operator !=**(right: Vector4)

Returns `true` if the vectors are not equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

Vector4 **operator ***(right: Projection)

Transforms (multiplies) the **Vector4** by the transpose of the given Projection matrix.

For transforming by inverse of a projection `projection.inverse() * vector` can be used instead. See Projection.inverse.

Vector4 **operator ***(right: Vector4)

Multiplies each component of the **Vector4** by the components of the given **Vector4**.

```
print(Vector4(10, 20, 30, 40) * Vector4(3, 4, 5, 6)) # Prints "(30, 80, 150, 240)"
```

Vector4 **operator ***(right: float)

Multiplies each component of the **Vector4** by the given float.

```
print(Vector4(10, 20, 30, 40) * 2) # Prints "(20, 40, 60, 80)"
```

Vector4 **operator ***(right: int)

Multiplies each component of the **Vector4** by the given int.

Vector4 **operator +**(right: Vector4)

Adds each component of the **Vector4** by the components of the given **Vector4**.

```
print(Vector4(10, 20, 30, 40) + Vector4(3, 4, 5, 6)) # Prints "(13, 24, 35, 46)"
```

Vector4 **operator -**(right: Vector4)

Subtracts each component of the **Vector4** by the components of the given **Vector4**.

```
print(Vector4(10, 20, 30, 40) - Vector4(3, 4, 5, 6)) # Prints "(7, 16, 25, 34)"
```

Vector4 **operator /**(right: Vector4)

Divides each component of the **Vector4** by the components of the given **Vector4**.

```
print(Vector4(10, 20, 30, 40) / Vector4(2, 5, 3, 4)) # Prints "(5, 4, 10, 10)"
```

Vector4 **operator /**(right: float)

Divides each component of the **Vector4** by the given float.

```
print(Vector4(10, 20, 30, 40) / 2 # Prints "(5, 10, 15, 20)"
```

Vector4 **operator /**(right: int)

Divides each component of the **Vector4** by the given int.

---

bool **operator <**(right: Vector4)

Compares two **Vector4** vectors by first checking if the X value of the left vector is less than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, Z values of the two vectors, and then with the W values. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator <=**(right: Vector4)

Compares two **Vector4** vectors by first checking if the X value of the left vector is less than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, Z values of the two vectors, and then with the W values. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator ==**(right: Vector4) 🔗

Returns `true` if the vectors are exactly equal.

Note: Due to floating-point precision errors, consider using is_equal_approx instead, which is more reliable.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator >**(right: Vector4) 🔗

Compares two **Vector4** vectors by first checking if the X value of the left vector is greater than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, Z values of the two vectors, and then with the W values. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

bool **operator >=**(right: Vector4) 🔗

Compares two **Vector4** vectors by first checking if the X value of the left vector is greater than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, Z values of the two vectors, and then with the W values. This operator is useful for sorting vectors.

Note: Vectors with @GDScript.NAN elements don't behave the same as other vectors. Therefore, the results from this operator may not be accurate if NaNs are included.

---

float **operator [\]**(index: int) 🔗

Access vector components using their `index`. `v[0]` is equivalent to `v.x`, `v[1]` is equivalent to `v.y`, `v[2]` is equivalent to `v.z`, and `v[3]` is equivalent to `v.w`.

Vector4 **operator unary+**() 🔗

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

Vector4 **operator unary-**() 🔗

Returns the negative value of the **Vector4**. This is the same as writing `Vector4(-v.x, -v.y, -v.z, -v.w)`. This operation flips the direction of the vector while keeping the same magnitude. With floats, the number zero can be either positive or negative.

Vector4i

A 4D vector using integer coordinates.

Description

A 4-element structure that can be used to represent 4D grid coordinates or any other quadruplet of integers.

It uses integer coordinates and is therefore preferable to Vector4 when exact precision is required. Note that the values are limited to 32 bits, and unlike Vector4 this cannot be configured with an engine build option. Use int or PackedInt64Array if 64-bit values are needed.

Note: In a boolean context, a Vector4i will evaluate to `false` if it's equal to `Vector4i(0, 0, 0, 0)`. Otherwise, a Vector4i will always evaluate to `true`.

Properties

int	w	`0`
int	x	`0`
int	y	`0`
int	z	`0`

Constructors

Vector4i	Vector4i()
Vector4i	Vector4i(from: Vector4i)
Vector4i	Vector4i(from: Vector4)
Vector4i	Vector4i(x: int, y: int, z: int, w: int)

Methods

Vector4i	abs() const
Vector4i	clamp(min: Vector4i, max: Vector4i) const
Vector4i	clampi(min: int, max: int) const
int	distance_squared_to(to: Vector4i) const
float	distance_to(to: Vector4i) const
float	length() const

int	length_squared() const
Vector4i	max(with: Vector4i) const
int	max_axis_index() const
Vector4i	maxi(with: int) const
Vector4i	min(with: Vector4i) const
int	min_axis_index() const
Vector4i	mini(with: int) const
Vector4i	sign() const
Vector4i	snapped(step: Vector4i) const
Vector4i	snappedi(step: int) const

Operators

bool	operator !=(right: Vector4i)

Vector4i	operator %(right: Vector4i)
Vector4i	operator %(right: int)
Vector4i	operator *(right: Vector4i)
Vector4	operator *(right: float)
Vector4i	operator *(right: int)
Vector4i	operator +(right: Vector4i)
Vector4i	operator -(right: Vector4i)
Vector4i	operator /(right: Vector4i)
Vector4	operator /(right: float)
Vector4i	operator /(right: int)
bool	operator <(right: Vector4i)
bool	operator <=(right: Vector4i)
bool	operator ==(right: Vector4i)

bool	operator >(right: Vector4i)
bool	operator >=(right: Vector4i)
int	operator [](index: int)
Vector4i	operator unary+()
Vector4i	operator unary-()

Constants

AXIS_X = `0`

Enumerated value for the X axis. Returned by max_axis_index and min_axis_index.

AXIS_Y = `1`

Enumerated value for the Y axis. Returned by max_axis_index and min_axis_index.

AXIS_Z = `2`

Enumerated value for the Z axis. Returned by max_axis_index and min_axis_index.

AXIS_W = `3`

Enumerated value for the W axis. Returned by max_axis_index and min_axis_index.

ZERO = `Vector4i(0, 0, 0, 0)` 🔗

Zero vector, a vector with all components set to `0`.

ONE = `Vector4i(1, 1, 1, 1)` 🔗

One vector, a vector with all components set to `1`.

MIN = `Vector4i(-2147483648, -2147483648, -2147483648, -2147483648)` 🔗

Min vector, a vector with all components equal to `INT32_MIN`. Can be used as a negative integer equivalent of Vector4.INF.

MAX = `Vector4i(2147483647, 2147483647, 2147483647, 2147483647)` 🔗

Max vector, a vector with all components equal to `INT32_MAX`. Can be used as an integer equivalent of Vector4.INF.

Property Descriptions

int **w** = `0` 🔗

The vector's W component. Also accessible by using the index position `[3]`.

int **x** = `0` 🔗

The vector's X component. Also accessible by using the index position `[0]`.

---

int **y** = `0`

The vector's Y component. Also accessible by using the index position `[1]`.

---

int **z** = `0`

The vector's Z component. Also accessible by using the index position `[2]`.

Constructor Descriptions

Vector4i **Vector4i**()

Constructs a default-initialized **Vector4i** with all components set to `0`.

Vector4i **Vector4i**(from: Vector4i)

Constructs a **Vector4i** as a copy of the given **Vector4i**.

Vector4i **Vector4i**(from: Vector4)

Constructs a new **Vector4i** from the given Vector4 by truncating components' fractional parts (rounding towards zero). For a different behavior consider passing the result of Vector4.ceil, Vector4.floor or Vector4.round to this constructor instead.

Vector4i **Vector4i**(x: int, y: int, z: int, w: int)

Returns a **Vector4i** with the given components.

Method Descriptions

Vector4i **abs**() const

Returns a new vector with all components in absolute values (i.e. positive).

Vector4i **clamp**(min: Vector4i, max: Vector4i) const

Returns a new vector with all components clamped between the components of `min` and `max`, by running @GlobalScope.clamp on each component.

Vector4i **clampi**(min: int, max: int) const

Returns a new vector with all components clamped between `min` and `max`, by running @GlobalScope.clamp on each component.

int **distance_squared_to**(to: Vector4i) const

Returns the squared distance between this vector and `to`.

This method runs faster than distance_to, so prefer it if you need to compare vectors or need the squared distance for some formula.

float **distance_to**(to: Vector4i) const

Returns the distance between this vector and `to`.

float **length**() const

Returns the length (magnitude) of this vector.

int **length_squared**() const

Returns the squared length (squared magnitude) of this vector.

This method runs faster than length, so prefer it if you need to compare vectors or need the squared distance for some formula.

Vector4i **max**(with: Vector4i) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector4i(maxi(x, with.x), maxi(y, with.y), maxi(z, with.z), maxi(w, with.w))`.

int **max_axis_index**() const

Returns the axis of the vector's highest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_X.

Vector4i **maxi**(with: int) const

Returns the component-wise maximum of this and `with`, equivalent to `Vector4i(maxi(x, with), maxi(y, with), maxi(z, with), maxi(w, with))`.

Vector4i **min**(with: Vector4i) const

Returns the component-wise minimum of this and `with`, equivalent to `Vector4i(mini(x, with.x), mini(y, with.y), mini(z, with.z), mini(w, with.w))`.

int **min_axis_index**() const 🔗

Returns the axis of the vector's lowest value. See `AXIS_*` constants. If all components are equal, this method returns AXIS_W.

Vector4i **mini**(with: int) const 🔗

Returns the component-wise minimum of this and `with`, equivalent to `Vector4i(mini(x, with), mini(y, with), mini(z, with), mini(w, with))`.

Vector4i **sign**() const 🔗

Returns a new vector with each component set to `1` if it's positive, `-1` if it's negative, and `0` if it's zero. The result is identical to calling @GlobalScope.sign on each component.

Vector4i **snapped**(step: Vector4i) const 🔗

Returns a new vector with each component snapped to the closest multiple of the corresponding component in `step`.

Vector4i **snappedi**(step: int) const 🔗

Returns a new vector with each component snapped to the closest multiple of `step`.

Operator Descriptions

bool **operator !=**(right: Vector4i) 🔗

Returns `true` if the vectors are not equal.

Vector4i **operator %**(right: Vector4i) 🔗

Gets the remainder of each component of the **Vector4i** with the components of the given **Vector4i**. This operation uses truncated division, which is often not desired as it does not work well with negative numbers. Consider using @GlobalScope.posmod instead if you want to handle negative numbers.

```
print(Vector4i(10, -20, 30, -40) % Vector4i(7, 8, 9, 10))  # Prints "(3, -4, 3, 0)"
```

Vector4i **operator %**(right: int) 🔗

Gets the remainder of each component of the **Vector4i** with the the given int. This operation uses truncated division, which is often not desired as it does not work well with negative numbers. Consider using @GlobalScope.posmod instead if you want to handle negative numbers.

```
print(Vector4i(10, -20, 30, -40) % 7)  # Prints "(3, -6, 2, -5)"
```

Vector4i **operator ***(right: Vector4i) 🔗

Multiplies each component of the **Vector4i** by the components of the given **Vector4i**.

```
print(Vector4i(10, 20, 30, 40) * Vector4i(3, 4, 5, 6)) # Prints
"(30, 80, 150, 240)"
```

Vector4 **operator ***(right: float)

Multiplies each component of the **Vector4i** by the given float.

Returns a Vector4 value due to floating-point operations.

```
print(Vector4i(10, 20, 30, 40) * 2) # Prints "(20, 40, 60, 80)"
```

Vector4i **operator ***(right: int)

Multiplies each component of the **Vector4i** by the given int.

Vector4i **operator +**(right: Vector4i)

Adds each component of the **Vector4i** by the components of the given **Vector4i**.

```
print(Vector4i(10, 20, 30, 40) + Vector4i(3, 4, 5, 6)) # Prints
"(13, 24, 35, 46)"
```

Vector4i **operator -**(right: Vector4i)

Subtracts each component of the **Vector4i** by the components of the given **Vector4i**.

```
print(Vector4i(10, 20, 30, 40) - Vector4i(3, 4, 5, 6)) # Prints
"(7, 16, 25, 34)"
```

Vector4i **operator /**(right: Vector4i)

Divides each component of the **Vector4i** by the components of the given **Vector4i**.

```
print(Vector4i(10, 20, 30, 40) / Vector4i(2, 5, 3, 4)) # Prints
"(5, 4, 10, 10)"
```

Vector4 **operator /**(right: float)

Divides each component of the **Vector4i** by the given float.

Returns a Vector4 value due to floating-point operations.

```
print(Vector4i(10, 20, 30, 40) / 2 # Prints "(5, 10, 15, 20)"
```

Vector4i **operator /**(right: int)

Divides each component of the **Vector4i** by the given int.

---

bool **operator <**(right: Vector4i)

Compares two **Vector4i** vectors by first checking if the X value of the left vector is less than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, Z values of the two vectors, and then with the W values. This operator is useful for sorting vectors.

---

bool **operator <=**(right: Vector4i)

Compares two **Vector4i** vectors by first checking if the X value of the left vector is less than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, Z values of the two vectors, and then with the W values. This operator is useful for sorting vectors.

bool **operator ==**(right: Vector4i)

Returns `true` if the vectors are exactly equal.

bool **operator >**(right: Vector4i)

Compares two **Vector4i** vectors by first checking if the X value of the left vector is greater than the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, Z values of the two vectors, and then with the W values. This operator is useful for sorting vectors.

bool **operator >=**(right: Vector4i)

Compares two **Vector4i** vectors by first checking if the X value of the left vector is greater than or equal to the X value of the `right` vector. If the X values are exactly equal, then it repeats this check with the Y values of the two vectors, Z values of the two vectors, and then with the W values. This operator is useful for sorting vectors.

int **operator []**(index: int)

Access vector components using their `index`. `v[0]` is equivalent to `v.x`, `v[1]` is equivalent to `v.y`, `v[2]` is equivalent to `v.z`, and `v[3]` is equivalent to `v.w`.

Vector4i **operator unary+**()

Returns the same value as if the `+` was not there. Unary `+` does nothing, but sometimes it can make your code more readable.

Vector4i **operator unary-()**

Returns the negative value of the **Vector4i**. This is the same as writing `Vector4i(-v.x, -v.y, -v.z, -v.w)`. This operation flips the direction of the vector while keeping the same magnitude.

Index

A | C | E | L | M | O | U

A

args (None attribute)

C

canvas (None attribute)
canvasResizePolicy (None attribute)

E

Engine() (class)
Engine.isWebGLAvailable() (Engine method)
Engine.load() (Engine method)
Engine.prototype.copyToFS() (Engine.prototype method)
Engine.prototype.init() (Engine.prototype method)
Engine.prototype.preloadFile() (Engine.prototype method)
Engine.prototype.requestQuit() (Engine.prototype method)
Engine.prototype.start() (Engine.prototype method)
Engine.prototype.startGame() (Engine.prototype method)
Engine.unload() (Engine method)
EngineConfig (None attribute)
executable (None attribute)

L

locale (None attribute)

M

mainPack (None attribute)

O

onExecute() (built-in function)
onExit() (built-in function)
onPrint() (built-in function)
onPrintError() (built-in function)
onProgress() (built-in function)

U

unloadAfterInit (None attribute)

Page not found

Sorry, we couldn't find that page. It may have been renamed or removed in the version of the documentation you're currently browsing.

If you're currently browsing the *latest* version of the documentation, try browsing the *stable* version of the documentation.

Alternatively, use the Search docs box on the left or go to the homepage.

@GDScript

Built-in GDScript constants, functions, and annotations.

Description

A list of GDScript-specific utility functions and annotations accessible from any script.

For the list of the global functions and constants see @GlobalScope.

Tutorials

- GDScript exports

Methods

Color	Color8(r8: int, g8: int, b8: int, a8: int = 255)
void	assert(condition: bool, message: String = "")
String	char(char: int)
Variant	convert(what: Variant, type: int)
Object	dict_to_inst(dictionary: Dictionary)

Array	get_stack()
Dictionary	inst_to_dict(instance: Object)
bool	is_instance_of(value: Variant, type: Variant)
int	len(var: Variant)
Resource	load(path: String)
Resource	preload(path: String)
void	print_debug(...) vararg
void	print_stack()
Array	range(...) vararg
bool	type_exists(type: StringName)

Constants

PI = `3.14159265358979`

Constant that represents how many times the diameter of a circle fits around its perimeter. This is equivalent to `TAU / 2`, or 180 degrees in rotations.

TAU = `6.28318530717959`

The circle constant, the circumference of the unit circle in radians. This is equivalent to `PI * 2`, or 360 degrees in rotations.

INF = `inf`

Positive floating-point infinity. This is the result of floating-point division when the divisor is `0.0`. For negative infinity, use `-INF`. Dividing by `-0.0` will result in negative infinity if the numerator is positive, so dividing by `0.0` is not the same as dividing by `-0.0` (despite `0.0 == -0.0` returning `true`).

Warning: Numeric infinity is only a concept with floating-point numbers, and has no equivalent for integers. Dividing an integer number by `0` will not result in INF and will result in a run-time error instead.

NAN = `nan`

"Not a Number", an invalid floating-point value. NAN has special properties, including that `!=` always returns `true`, while other comparison operators always return `false`. This is true even when comparing with itself (`NAN == NAN` returns `false` and `NAN != NAN` returns `true`). It is returned by some invalid operations, such as dividing floating-point `0.0` by `0.0`.

Warning: "Not a Number" is only a concept with floating-point numbers, and has no equivalent for integers. Dividing an integer `0` by `0` will not result in NAN and will result in a run-time error instead.

Annotations

@export()

Mark the following property as exported (editable in the Inspector dock and saved to disk). To control the type of the exported property, use the type hint notation.

```
extends Node

enum Direction {LEFT, RIGHT, UP, DOWN}

# Built-in types.
@export var string = ""
@export var int_number = 5
@export var float_number: float = 5

# Enums.
@export var type: Variant.Type
@export var format: Image.Format
@export var direction: Direction

# Resources.
@export var image: Image
@export var custom_resource: CustomResource

# Nodes.
@export var node: Node
@export var custom_node: CustomNode

# Typed arrays.
@export var int_array: Array[int]
@export var direction_array: Array[Direction]
@export var image_array: Array[Image]
@export var node_array: Array[Node]
```

Note: Custom resources and nodes should be registered as global classes using `class_name`, since the Inspector currently only supports global classes. Otherwise, a less specific type will be exported instead.

Note: Node export is only supported in Node-derived classes and has a number of other limitations.

@export_category(name: String)

Define a new category for the following exported properties. This helps to organize properties in the Inspector dock.

See also @GlobalScope.PROPERTY_USAGE_CATEGORY.

```
@export_category("Statistics")
@export var hp = 30
@export var speed = 1.25
```

Note: Categories in the Inspector dock's list usually divide properties coming from different classes (Node, Node2D, Sprite, etc.). For better clarity, it's recommended to use @export_group and @export_subgroup, instead.

@export_color_no_alpha()

Export a Color, Array[Color], or PackedColorArray property without allowing its transparency (Color.a) to be edited.

See also @GlobalScope.PROPERTY_HINT_COLOR_NO_ALPHA.

```
@export_color_no_alpha var dye_color: Color
@export_color_no_alpha var dye_colors: Array[Color]
```

@export_custom(hint: PropertyHint, hint_string: String, usage: BitField[PropertyUsageFlags] = 6)

Allows you to set a custom hint, hint string, and usage flags for the exported property. Note that there's no validation done in GDScript, it will just pass the parameters to the editor.

```
@export_custom(PROPERTY_HINT_NONE, "suffix:m") var suffix:
Vector3
```

Note: Regardless of the `usage` value, the @GlobalScope.PROPERTY_USAGE_SCRIPT_VARIABLE flag is always added, as with any explicitly declared script variable.

@export_dir()

Export a String, Array[String], or PackedStringArray property as a path to a directory. The path will be limited to the project folder and its subfolders. See @export_global_dir to allow picking from the entire filesystem.

See also @GlobalScope.PROPERTY_HINT_DIR.

```
@export_dir var sprite_folder_path: String
@export_dir var sprite_folder_paths: Array[String]
```

@export_enum(names: String, ...) vararg

Export an int, String, Array[int], Array[String], PackedByteArray, PackedInt32Array, PackedInt64Array, or PackedStringArray property as an enumerated list of options (or an array of options). If the property is an int, then the index of the value is stored, in the same order the values are provided. You can add explicit values using a colon. If the property is a String, then the value is stored.

See also @GlobalScope.PROPERTY_HINT_ENUM.

```
@export_enum("Warrior", "Magician", "Thief") var
character_class: int
@export_enum("Slow:30", "Average:60", "Very Fast:200") var
character_speed: int
@export_enum("Rebecca", "Mary", "Leah") var character_name:
String
```

```
@export_enum("Sword", "Spear", "Mace") var character_items:
Array[int]
@export_enum("double_jump", "climb", "dash") var
character_skills: Array[String]
```

If you want to set an initial value, you must specify it explicitly:

```
@export_enum("Rebecca", "Mary", "Leah") var character_name:
String = "Rebecca"
```

If you want to use named GDScript enums, then use @export instead:

```
enum CharacterName {REBECCA, MARY, LEAH}
@export var character_name: CharacterName

enum CharacterItem {SWORD, SPEAR, MACE}
@export var character_items: Array[CharacterItem]
```

@export_exp_easing(hints: String = "", ...) vararg

Export a floating-point property with an easing editor widget. Additional hints can be provided to adjust the behavior of the widget. `"attenuation"` flips the curve, which makes it more intuitive for editing attenuation properties. `"positive_only"` limits values to only be greater than or equal to zero.

See also @GlobalScope.PROPERTY_HINT_EXP_EASING.

```
@export_exp_easing var transition_speed
@export_exp_easing("attenuation") var fading_attenuation
@export_exp_easing("positive_only") var effect_power
@export_exp_easing var speeds: Array[float]
```

@export_file(filter: String = "", ...) vararg

Export a String, Array[String], or PackedStringArray property as a path to a file. The path will be limited to the project folder and its subfolders. See @export_global_file to allow picking from the entire filesystem.

If `filter` is provided, only matching files will be available for picking.

See also @GlobalScope.PROPERTY_HINT_FILE.

```
@export_file var sound_effect_path: String
@export_file("*.txt") var notes_path: String
@export_file var level_paths: Array[String]
```

@export_flags(names: String, ...) vararg

Export an integer property as a bit flag field. This allows to store several "checked" or `true` values with one property, and comfortably select them from the Inspector dock.

See also @GlobalScope.PROPERTY_HINT_FLAGS.

```
@export_flags("Fire", "Water", "Earth", "Wind") var
spell_elements = 0
```

You can add explicit values using a colon:

```
@export_flags("Self:4", "Allies:8", "Foes:16") var
spell_targets = 0
```

You can also combine several flags:

```
@export_flags("Self:4", "Allies:8", "Self and Allies:12",
"Foes:16")
var spell_targets = 0
```

Note: A flag value must be at least `1` and at most `2 ** 32 - 1`.

Note: Unlike @export_enum, the previous explicit value is not taken into account. In the following example, A is 16, B is 2, C is 4.

```
@export_flags("A:16", "B", "C") var x
```

You can also use the annotation on Array[int], PackedByteArray, PackedInt32Array, and PackedInt64Array

```
@export_flags("Fire", "Water", "Earth", "Wind") var
phase_elements: Array[int]
```

@export_flags_2d_navigation()

Export an integer property as a bit flag field for 2D navigation layers. The widget in the Inspector dock will use the layer names defined in ProjectSettings.layer_names/2d_navigation/layer_1.

See also @GlobalScope.PROPERTY_HINT_LAYERS_2D_NAVIGATION.

```
@export_flags_2d_navigation var navigation_layers: int
@export_flags_2d_navigation var navigation_layers_array:
Array[int]
```

@export_flags_2d_physics()

Export an integer property as a bit flag field for 2D physics layers. The widget in the Inspector dock will use the layer names defined in ProjectSettings.layer_names/2d_physics/layer_1.

See also @GlobalScope.PROPERTY_HINT_LAYERS_2D_PHYSICS.

```
@export_flags_2d_physics var physics_layers: int
@export_flags_2d_physics var physics_layers_array: Array[int]
```

@export_flags_2d_render()

Export an integer property as a bit flag field for 2D render layers. The widget in the Inspector dock will use the layer names defined in ProjectSettings.layer_names/2d_render/layer_1.

See also @GlobalScope.PROPERTY_HINT_LAYERS_2D_RENDER.

```
@export_flags_2d_render var render_layers: int
@export_flags_2d_render var render_layers_array: Array[int]
```

@export_flags_3d_navigation() 🔗

Export an integer property as a bit flag field for 3D navigation layers. The widget in the Inspector dock will use the layer names defined in ProjectSettings.layer_names/3d_navigation/layer_1.

See also @GlobalScope.PROPERTY_HINT_LAYERS_3D_NAVIGATION.

```
@export_flags_3d_navigation var navigation_layers: int
@export_flags_3d_navigation var navigation_layers_array:
Array[int]
```

@export_flags_3d_physics() 🔗

Export an integer property as a bit flag field for 3D physics layers. The widget in the Inspector dock will use the layer names defined in ProjectSettings.layer_names/3d_physics/layer_1.

See also @GlobalScope.PROPERTY_HINT_LAYERS_3D_PHYSICS.

```
@export_flags_3d_physics var physics_layers: int
@export_flags_3d_physics var physics_layers_array: Array[int]
```

@export_flags_3d_render() 🔗

Export an integer property as a bit flag field for 3D render layers. The widget in the Inspector dock will use the layer names defined in

ProjectSettings.layer_names/3d_render/layer_1.

See also @GlobalScope.PROPERTY_HINT_LAYERS_3D_RENDER.

```
@export_flags_3d_render var render_layers: int
@export_flags_3d_render var render_layers_array: Array[int]
```

@export_flags_avoidance()

Export an integer property as a bit flag field for navigation avoidance layers. The widget in the Inspector dock will use the layer names defined in ProjectSettings.layer_names/avoidance/layer_1.

See also @GlobalScope.PROPERTY_HINT_LAYERS_AVOIDANCE.

```
@export_flags_avoidance var avoidance_layers: int
@export_flags_avoidance var avoidance_layers_array: Array[int]
```

@export_global_dir()

Export a String, Array[String], or PackedStringArray property as an absolute path to a directory. The path can be picked from the entire filesystem. See @export_dir to limit it to the project folder and its subfolders.

See also @GlobalScope.PROPERTY_HINT_GLOBAL_DIR.

```
@export_global_dir var sprite_folder_path: String
@export_global_dir var sprite_folder_paths: Array[String]
```

@export_global_file(filter: String = "", ...) vararg

Export a String, Array[String], or PackedStringArray property as an absolute path to a file. The path can be picked from the entire filesystem. See @export_file to limit it to the project folder and its subfolders.

If `filter` is provided, only matching files will be available for picking.

See also @GlobalScope.PROPERTY_HINT_GLOBAL_FILE.

```
@export_global_file var sound_effect_path: String
@export_global_file("*.txt") var notes_path: String
@export_global_file var multiple_paths: Array[String]
```

@export_group(name: String, prefix: String = "") 🔗

Define a new group for the following exported properties. This helps to organize properties in the Inspector dock. Groups can be added with an optional `prefix`, which would make group to only consider properties that have this prefix. The grouping will break on the first property that doesn't have a prefix. The prefix is also removed from the property's name in the Inspector dock.

If no `prefix` is provided, then every following property will be added to the group. The group ends when then next group or category is defined. You can also force end a group by using this annotation with empty strings for parameters, `@export_group("", "")`.

Groups cannot be nested, use @export_subgroup to add subgroups within groups.

See also @GlobalScope.PROPERTY_USAGE_GROUP.

```
@export_group("Racer Properties")
@export var nickname = "Nick"
@export var age = 26

@export_group("Car Properties", "car_")
@export var car_label = "Speedy"
@export var car_number = 3

@export_group("", "")
@export var ungrouped_number = 3
```

@export_multiline() 🔗

Export a String, Array[String], PackedStringArray, Dictionary or Array[Dictionary] property with a large TextEdit widget instead of a LineEdit. This adds support for multiline content and makes it easier to edit large amount of text stored in the property.

See also @GlobalScope.PROPERTY_HINT_MULTILINE_TEXT.

```
@export_multiline var character_biography
@export_multiline var npc_dialogs: Array[String]
```

@export_node_path(type: String = "", ...) vararg 🔗

Export a NodePath or Array[NodePath] property with a filter for allowed node types.

See also @GlobalScope.PROPERTY_HINT_NODE_PATH_VALID_TYPES.

```
@export_node_path("Button", "TouchScreenButton") var
some_button
@export_node_path("Button", "TouchScreenButton") var
many_buttons: Array[NodePath]
```

Note: The type must be a native class or a globally registered script (using the `class_name` keyword) that inherits Node.

@export_placeholder(placeholder: String) 🔗

Export a String, Array[String], or PackedStringArray property with a placeholder text displayed in the editor widget when no value is present.

See also @GlobalScope.PROPERTY_HINT_PLACEHOLDER_TEXT.

```
@export_placeholder("Name in lowercase") var character_id:
String
@export_placeholder("Name in lowercase") var friend_ids:
Array[String]
```

@export_range(min: float, max: float, step: float = 1.0, extra_hints: String = "", ...) vararg

Export an int, float, Array[int], Array[float], PackedByteArray, PackedInt32Array, PackedInt64Array, PackedFloat32Array, or PackedFloat64Array property as a range value. The range must be defined by `min` and `max`, as well as an optional `step` and a variety of extra hints. The `step` defaults to `1` for integer properties. For floating-point numbers this value depends on your EditorSettings.interface/inspector/default_float_step setting.

If hints `"or_greater"` and `"or_less"` are provided, the editor widget will not cap the value at range boundaries. The `"exp"` hint will make the edited values on range to change exponentially. The `"hide_slider"` hint will hide the slider element of the editor widget.

Hints also allow to indicate the units for the edited value. Using `"radians_as_degrees"` you can specify that the actual value is in radians, but should be displayed in degrees in the Inspector dock (the range values are also in degrees). `"degrees"` allows to add a degree sign as a unit suffix (the value is unchanged). Finally, a custom suffix can be provided using `"suffix:unit"`, where "unit" can be any string.

See also @GlobalScope.PROPERTY_HINT_RANGE.

```
@export_range(0, 20) var number
@export_range(-10, 20) var number
@export_range(-10, 20, 0.2) var number: float
@export_range(0, 20) var numbers: Array[float]

@export_range(0, 100, 1, "or_greater") var power_percent
```

```
@export_range(0, 100, 1, "or_greater", "or_less") var
health_delta

@export_range(-180, 180, 0.001, "radians_as_degrees") var
angle_radians
@export_range(0, 360, 1, "degrees") var angle_degrees
@export_range(-8, 8, 2, "suffix:px") var target_offset
```

@export_storage()

Export a property with [@GlobalScope.PROPERTY_USAGE_STORAGE](#) flag. The property is not displayed in the editor, but it is serialized and stored in the scene or resource file. This can be useful for [@tool](#) scripts. Also the property value is copied when [Resource.duplicate](#) or [Node.duplicate](#) is called, unlike non-exported variables.

```
var a # Not stored in the file, not displayed in the editor.
@export_storage var b # Stored in the file, not displayed in
the editor.
@export var c: int # Stored in the file, displayed in the
editor.
```

@export_subgroup(name: String, prefix: String = "")

Define a new subgroup for the following exported properties. This helps to organize properties in the Inspector dock. Subgroups work exactly like groups, except they need a parent group to exist. See [@export_group](#).

See also [@GlobalScope.PROPERTY_USAGE_SUBGROUP](#).

```
@export_group("Racer Properties")
@export var nickname = "Nick"
@export var age = 26

@export_subgroup("Car Properties", "car_")
```

```
@export var car_label = "Speedy"
@export var car_number = 3
```

Note: Subgroups cannot be nested, they only provide one extra level of depth. Just like the next group ends the previous group, so do the subsequent subgroups.

@icon(icon_path: String) 🔗

Add a custom icon to the current script. The icon specified at `icon_path` is displayed in the Scene dock for every node of that class, as well as in various editor dialogs.

```
@icon("res://path/to/class/icon.svg")
```

Note: Only the script can have a custom icon. Inner classes are not supported.

Note: As annotations describe their subject, the @icon annotation must be placed before the class definition and inheritance.

Note: Unlike other annotations, the argument of the @icon annotation must be a string literal (constant expressions are not supported).

@onready() 🔗

Mark the following property as assigned when the Node is ready. Values for these properties are not assigned immediately when the node is initialized (Object._init), and instead are computed and stored right before Node._ready.

```
@onready var character_name: Label = $Label
```

@rpc(mode: String = "authority", sync: String = "call_remote", transfer_mode: String = "unreliable", transfer_channel: int = 0) 🔗

Mark the following method for remote procedure calls. See High-level multiplayer.

If `mode` is set as `"any_peer"`, allows any peer to call this RPC function. Otherwise, only the authority peer is allowed to call it and `mode` should be kept as `"authority"`. When configuring functions as RPCs with Node.rpc_config, each of these modes respectively corresponds to the MultiplayerAPI.RPC_MODE_AUTHORITY and MultiplayerAPI.RPC_MODE_ANY_PEER RPC modes. See RPCMode. If a peer that is not the authority tries to call a function that is only allowed for the authority, the function will not be executed. If the error can be detected locally (when the RPC configuration is consistent between the local and the remote peer), an error message will be displayed on the sender peer. Otherwise, the remote peer will detect the error and print an error there.

If `sync` is set as `"call_remote"`, the function will only be executed on the remote peer, but not locally. To run this function locally too, set `sync` to `"call_local"`. When configuring functions as RPCs with Node.rpc_config, this is equivalent to setting `call_local` to `true`.

The `transfer_mode` accepted values are `"unreliable"`, `"unreliable_ordered"`, or `"reliable"`. It sets the transfer mode of the underlying MultiplayerPeer. See MultiplayerPeer.transfer_mode.

The `transfer_channel` defines the channel of the underlying MultiplayerPeer. See MultiplayerPeer.transfer_channel.

The order of `mode`, `sync` and `transfer_mode` does not matter, but values related to the same argument must not be used more than once. `transfer_channel` always has to be the 4th argument (you must specify 3 preceding arguments).

```
@rpc
func fn(): pass

@rpc("any_peer", "unreliable_ordered")
func fn_update_pos(): pass

@rpc("authority", "call_remote", "unreliable", 0) # Equivalent
to @rpc
func fn_default(): pass
```

@static_unload()

Make a script with static variables to not persist after all references are lost. If the script is loaded again the static variables will revert to their default values.

Note: As annotations describe their subject, the @static_unload annotation must be placed before the class definition and inheritance.

Warning: Currently, due to a bug, scripts are never freed, even if @static_unload annotation is used.

@tool()

Mark the current script as a tool script, allowing it to be loaded and executed by the editor. See Running code in the editor.

```
@tool
extends Node
```

Note: As annotations describe their subject, the @tool annotation must be placed before the class definition and inheritance.

@warning_ignore(warning: String, ...) vararg

Mark the following statement to ignore the specified `warning`. See GDScript warning system.

```
func test():
    print("hello")
    return
    @warning_ignore("unreachable_code")
    print("unreachable")
```

Method Descriptions

Color **Color8**(r8: int, g8: int, b8: int, a8: int = 255)

Returns a Color constructed from red (`r8`), green (`g8`), blue (`b8`), and optionally alpha (`a8`) integer channels, each divided by `255.0` for their final value. Using Color8 instead of the standard Color constructor is useful when you need to match exact color values in an Image.

```
var red = Color8(255, 0, 0)             # Same as Color(1, 0, 0).
var dark_blue = Color8(0, 0, 51)        # Same as Color(0, 0, 0.2).
var my_color = Color8(306, 255, 0, 102) # Same as Color(1.2, 1, 0, 0.4).
```

Note: Due to the lower precision of Color8 compared to the standard Color constructor, a color created with Color8 will generally not be equal to the same color created with the standard Color constructor. Use Color.is_equal_approx for comparisons to avoid issues with floating-point precision error.

---

void **assert**(condition: bool, message: String = "")

Asserts that the `condition` is `true`. If the `condition` is `false`, an error is generated. When running from the editor, the running project will also be paused until you resume it. This can be used as a stronger form of @GlobalScope.push_error for reporting errors to project developers or add-on users.

An optional `message` can be shown in addition to the generic "Assertion failed" message. You can use this to provide additional details about why the assertion failed.

Warning: For performance reasons, the code inside assert is only executed in debug builds or when running the project from the editor. Don't include code that has side effects in an assert call. Otherwise, the project will behave differently when exported in release mode.

```
# Imagine we always want speed to be between 0 and 20.
var speed = -10
assert(speed < 20) # True, the program will continue.
assert(speed >= 0) # False, the program will stop.
assert(speed >= 0 and speed < 20) # You can also combine the two conditional statements in one check.
assert(speed < 20, "the speed limit is 20") # Show a message.
```

Note: assert is a keyword, not a function. So you cannot access it as a Callable or use it inside expressions.

String **char**(char: int) 🔗

Returns a single character (as a String) of the given Unicode code point (which is compatible with ASCII code).

```
a = char(65)      # a is "A"
a = char(65 + 32) # a is "a"
a = char(8364)    # a is "€"
```

Variant **convert**(what: Variant, type: int)

Deprecated: Use @GlobalScope.type_convert instead.

Converts `what` to `type` in the best way possible. The `type` uses the Variant.Type values.

```
var a = [4, 2.5, 1.2]
print(a is Array) # Prints true

var b = convert(a, TYPE_PACKED_BYTE_ARRAY)
print(b)          # Prints [4, 2, 1]
print(b is Array) # Prints false
```

Object **dict_to_inst**(dictionary: Dictionary)

Converts a `dictionary` (created with inst_to_dict) back to an Object instance. Can be useful for deserializing.

Array **get_stack**()

Returns an array of dictionaries representing the current call stack. See also print_stack.

```
func _ready():
    foo()

func foo():
    bar()

func bar():
    print(get_stack())
```

Starting from `_ready()`, `bar()` would print:

```
[{function:bar, line:12, source:res://script.gd},
{function:foo, line:9, source:res://script.gd},
{function:_ready, line:6, source:res://script.gd}]
```

Note: This function only works if the running instance is connected to a debugging server (i.e. an editor instance). get_stack will not work in projects exported in release mode, or in projects exported in debug mode if not connected to a debugging server.

Note: Calling this function from a Thread is not supported. Doing so will return an empty array.

Dictionary **inst_to_dict**(instance: Object) 🔗

Returns the passed `instance` converted to a Dictionary. Can be useful for serializing.

Note: Cannot be used to serialize objects with built-in scripts attached or objects allocated within built-in scripts.

```
var foo = "bar"
func _ready():
    var d = inst_to_dict(self)
    print(d.keys())
    print(d.values())
```

Prints out:

```
[@subpath, @path, foo]
[, res://test.gd, bar]
```

---

bool **is_instance_of**(value: Variant, type: Variant) 🔗

Returns `true` if `value` is an instance of `type`. The `type` value must be one of the following:

- A constant from the Variant.Type enumeration, for example @GlobalScope.TYPE_INT.

- An Object-derived class which exists in ClassDB, for example Node.
- A Script (you can use any class, including inner one).

Unlike the right operand of the `is` operator, `type` can be a non-constant value. The `is` operator supports more features (such as typed arrays). Use the operator instead of this method if you do not need dynamic type checking.

Examples:

```
print(is_instance_of(a, TYPE_INT))
print(is_instance_of(a, Node))
print(is_instance_of(a, MyClass))
print(is_instance_of(a, MyClass.InnerClass))
```

Note: If `value` and/or `type` are freed objects (see @GlobalScope.is_instance_valid), or `type` is not one of the above options, this method will raise a runtime error.

See also @GlobalScope.typeof, type_exists, Array.is_same_typed (and other Array methods).

---

int **len**(var: Variant) 🔗

Returns the length of the given Variant `var`. The length can be the character count of a String or StringName, the element count of any array type, or the size of a Dictionary. For every other Variant type, a run-time error is generated and execution is stopped.

```
a = [1, 2, 3, 4]
len(a) # Returns 4

b = "Hello!"
len(b) # Returns 6
```

Resource **load**(path: String) 🔗

Returns a Resource from the filesystem located at the absolute `path`. Unless it's already referenced elsewhere (such as in another script or in the scene), the resource is loaded from disk on function call, which might cause a slight delay, especially when loading large scenes. To avoid unnecessary delays when loading something multiple times, either store the resource in a variable or use preload. This method is equivalent of using ResourceLoader.load with ResourceLoader.CACHE_MODE_REUSE.

Note: Resource paths can be obtained by right-clicking on a resource in the FileSystem dock and choosing "Copy Path", or by dragging the file from the FileSystem dock into the current script.

```
# Load a scene called "main" located in the root of the project directory and cache it in a variable.
var main = load("res://main.tscn") # main will contain a PackedScene resource.
```

Important: Relative paths are *not* relative to the script calling this method, instead it is prefixed with `"res://"`. Loading from relative paths might not work as expected.

This function is a simplified version of ResourceLoader.load, which can be used for more advanced scenarios.

Note: Files have to be imported into the engine first to load them using this function. If you want to load Images at run-time, you may use Image.load. If you want to import audio files, you can use the snippet described in AudioStreamMP3.data.

Note: If ProjectSettings.editor/export/convert_text_resources_to_binary is `true`, load will not be able to read converted files in an exported project. If you rely on run-time loading of files present within the PCK, set

ProjectSettings.editor/export/convert_text_resources_to_binary to `false`.

Resource **preload**(path: String)

Returns a Resource from the filesystem located at `path`. During run-time, the resource is loaded when the script is being parsed. This function effectively acts as a reference to that resource. Note that this function requires `path` to be a constant String. If you want to load a resource from a dynamic/variable path, use load.

Note: Resource paths can be obtained by right-clicking on a resource in the Assets Panel and choosing "Copy Path", or by dragging the file from the FileSystem dock into the current script.

```
# Create instance of a scene.
var diamond = preload("res://diamond.tscn").instantiate()
```

Note: preload is a keyword, not a function. So you cannot access it as a Callable.

---

void **print_debug**(...) vararg

Like @GlobalScope.print, but includes the current stack frame when running with the debugger turned on.

The output in the console may look like the following:

```
Test print
At: res://test.gd:15:_process()
```

Note: Calling this function from a Thread is not supported. Doing so will instead print the thread ID.

---

void **print_stack**()

Prints a stack trace at the current code location. See also get_stack.

The output in the console may look like the following:

```
Frame 0 - res://test.gd:16 in function '_process'
```

Note: This function only works if the running instance is connected to a debugging server (i.e. an editor instance). print_stack will not work in projects exported in release mode, or in projects exported in debug mode if not connected to a debugging server.

Note: Calling this function from a Thread is not supported. Doing so will instead print the thread ID.

Array **range**(...) vararg

Returns an array with the given range. range can be called in three ways:

`range(n: int)`: Starts from 0, increases by steps of 1, and stops *before* `n`. The argument `n` is **exclusive**.

`range(b: int, n: int)`: Starts from `b`, increases by steps of 1, and stops *before* `n`. The arguments `b` and `n` are **inclusive** and **exclusive**, respectively.

`range(b: int, n: int, s: int)`: Starts from `b`, increases/decreases by steps of `s`, and stops *before* `n`. The arguments `b` and `n` are **inclusive** and **exclusive**, respectively. The argument `s` **can** be negative, but not `0`. If `s` is `0`, an error message is printed.

range converts all arguments to int before processing.

Note: Returns an empty array if no value meets the value constraint (e.g. `range(2, 5, -1)` or `range(5, 5, 1)`).

Examples:

```
print(range(4))         # Prints [0, 1, 2, 3]
print(range(2, 5))      # Prints [2, 3, 4]
print(range(0, 6, 2))   # Prints [0, 2, 4]
print(range(4, 1, -1)) # Prints [4, 3, 2]
```

To iterate over an Array backwards, use:

```
var array = [3, 6, 9]
for i in range(array.size() - 1, -1, -1):
    print(array[i])
```

Output:

```
9
6
3
```

To iterate over float, convert them in the loop.

```
for i in range (3, 0, -1):
    print(i / 10.0)
```

Output:

```
0.3
0.2
0.1
```

---

bool **type_exists**(type: StringName) 🔗

Returns `true` if the given Object-derived class exists in ClassDB. Note that Variant data types are not registered in ClassDB.

```
type_exists("Sprite2D") # Returns true
type_exists("NonExistentClass") # Returns false
```

@GlobalScope

Global scope constants and functions.

Description

A list of global scope enumerated constants and built-in functions. This is all that resides in the globals, constants regarding error codes, keycodes, property hints, etc.

Singletons are also documented here, since they can be accessed from anywhere.

For the entries related to GDScript which can be accessed in any script see @GDScript.

Note

There are notable differences when using this API with C#. See C# API differences to GDScript for more information.

Tutorials

- Random number generation

Properties

AudioServer	AudioServer

CameraServer	CameraServer
ClassDB	ClassDB
DisplayServer	DisplayServer
EditorInterface	EditorInterface
Engine	Engine
EngineDebugger	EngineDebugger
GDExtensionManager	GDExtensionManager
Geometry2D	Geometry2D
Geometry3D	Geometry3D
IP	IP
Input	Input
InputMap	InputMap
JavaClassWrapper	JavaClassWrapper

JavaScriptBridge	JavaScriptBridge
Marshalls	Marshalls
NativeMenu	NativeMenu
NavigationMeshGenerator	NavigationMeshGenerator
NavigationServer2D	NavigationServer2D
NavigationServer3D	NavigationServer3D
OS	OS
Performance	Performance
PhysicsServer2D	PhysicsServer2D
PhysicsServer2DManager	PhysicsServer2DManager
PhysicsServer3D	PhysicsServer3D
PhysicsServer3DManager	PhysicsServer3DManager
ProjectSettings	ProjectSettings

RenderingServer	RenderingServer
ResourceLoader	ResourceLoader
ResourceSaver	ResourceSaver
ResourceUID	ResourceUID
TextServerManager	TextServerManager
ThemeDB	ThemeDB
Time	Time
TranslationServer	TranslationServer
WorkerThreadPool	WorkerThreadPool
XRServer	XRServer

Methods

Variant	abs(x: Variant)

float	absf(x: float)
int	absi(x: int)
float	acos(x: float)
float	acosh(x: float)
float	angle_difference(from: float, to: float)
float	asin(x: float)
float	asinh(x: float)
float	atan(x: float)
float	atan2(y: float, x: float)
float	atanh(x: float)
float	bezier_derivative(start: float, control_1: float, control_2: float, end: float, t: float)
float	bezier_interpolate(start: float, control_1: float, control_2: float, end: float, t: float)

Variant	bytes_to_var(bytes: PackedByteArray)
Variant	bytes_to_var_with_objects(bytes: PackedByteArray)
Variant	ceil(x: Variant)
float	ceilf(x: float)
int	ceili(x: float)
Variant	clamp(value: Variant, min: Variant, max: Variant)
float	clampf(value: float, min: float, max: float)
int	clampi(value: int, min: int, max: int)
float	cos(angle_rad: float)
float	cosh(x: float)
float	cubic_interpolate(from: float, to: float, pre: float, post: float, weight: float)

float	cubic_interpolate_angle(from: float, to: float, pre: float, post: float, weight: float)
float	cubic_interpolate_angle_in_time(from: float, to: float, pre: float, post: float, weight: float, to_t: float, pre_t: float, post_t: float)
float	cubic_interpolate_in_time(from: float, to: float, pre: float, post: float, weight: float, to_t: float, pre_t: float, post_t: float)
float	db_to_linear(db: float)
float	deg_to_rad(deg: float)
float	ease(x: float, curve: float)
String	error_string(error: int)
float	exp(x: float)
Variant	floor(x: Variant)
float	floorf(x: float)
int	floori(x: float)

float	fmod(x: float, y: float)
float	fposmod(x: float, y: float)
int	hash(variable: Variant)
Object	instance_from_id(instance_id: int)
float	inverse_lerp(from: float, to: float, weight: float)
bool	is_equal_approx(a: float, b: float)
bool	is_finite(x: float)
bool	is_inf(x: float)
bool	is_instance_id_valid(id: int)
bool	is_instance_valid(instance: Variant)
bool	is_nan(x: float)
bool	is_same(a: Variant, b: Variant)
bool	is_zero_approx(x: float)

Variant	lerp(from: Variant, to: Variant, weight: Variant)
float	lerp_angle(from: float, to: float, weight: float)
float	lerpf(from: float, to: float, weight: float)
float	linear_to_db(lin: float)
float	log(x: float)
Variant	max(...) vararg
float	maxf(a: float, b: float)
int	maxi(a: int, b: int)
Variant	min(...) vararg
float	minf(a: float, b: float)
int	mini(a: int, b: int)
float	move_toward(from: float, to: float, delta: float)
int	nearest_po2(value: int)

float	pingpong(value: float, length: float)
int	posmod(x: int, y: int)
float	pow(base: float, exp: float)
void	print(...) vararg
void	print_rich(...) vararg
void	print_verbose(...) vararg
void	printerr(...) vararg
void	printraw(...) vararg
void	prints(...) vararg
void	printt(...) vararg
void	push_error(...) vararg
void	push_warning(...) vararg
float	rad_to_deg(rad: float)

PackedInt64Array	rand_from_seed(seed: int)
float	randf()
float	randf_range(from: float, to: float)
float	randfn(mean: float, deviation: float)
int	randi()
int	randi_range(from: int, to: int)
void	randomize()
float	remap(value: float, istart: float, istop: float, ostart: float, ostop: float)
int	rid_allocate_id()
RID	rid_from_int64(base: int)
float	rotate_toward(from: float, to: float, delta: float)
Variant	round(x: Variant)

float	roundf(x: float)
int	roundi(x: float)
void	seed(base: int)
Variant	sign(x: Variant)
float	signf(x: float)
int	signi(x: int)
float	sin(angle_rad: float)
float	sinh(x: float)
float	smoothstep(from: float, to: float, x: float)
Variant	snapped(x: Variant, step: Variant)
float	snappedf(x: float, step: float)
int	snappedi(x: float, step: int)
float	sqrt(x: float)

int	step_decimals(x: float)
String	str(...) vararg
Variant	str_to_var(string: String)
float	tan(angle_rad: float)
float	tanh(x: float)
Variant	type_convert(variant: Variant, type: int)
String	type_string(type: int)
int	typeof(variable: Variant)
PackedByteArray	var_to_bytes(variable: Variant)
PackedByteArray	var_to_bytes_with_objects(variable: Variant)
String	var_to_str(variable: Variant)
Variant	weakref(obj: Variant)
Variant	wrap(value: Variant, min: Variant, max: Variant)

float	wrapf(value: float, min: float, max: float)
int	wrapi(value: int, min: int, max: int)

Enumerations

enum **Side**: 🔗

Side **SIDE_LEFT** = `0`

Left side, usually used for Control or StyleBox-derived classes.

Side **SIDE_TOP** = `1`

Top side, usually used for Control or StyleBox-derived classes.

Side **SIDE_RIGHT** = `2`

Right side, usually used for Control or StyleBox-derived classes.

Side **SIDE_BOTTOM** = `3`

Bottom side, usually used for Control or StyleBox-derived classes.

enum **Corner**: 🔗

Corner **CORNER_TOP_LEFT** = `0`

Top-left corner.

Corner **CORNER_TOP_RIGHT** = `1`

Top-right corner.

Corner **CORNER_BOTTOM_RIGHT** = `2`

Bottom-right corner.

Corner **CORNER_BOTTOM_LEFT** = `3`

Bottom-left corner.

---

enum **Orientation**:

Orientation **VERTICAL** = `1`

General vertical alignment, usually used for Separator, ScrollBar, Slider, etc.

Orientation **HORIZONTAL** = `0`

General horizontal alignment, usually used for Separator, ScrollBar, Slider, etc.

---

enum **ClockDirection**:

ClockDirection **CLOCKWISE** = `0`

Clockwise rotation. Used by some methods (e.g. Image.rotate_90).

ClockDirection **COUNTERCLOCKWISE** = `1`

Counter-clockwise rotation. Used by some methods (e.g. Image.rotate_90).

---

enum **HorizontalAlignment**:

HorizontalAlignment **HORIZONTAL_ALIGNMENT_LEFT** = `0`

Horizontal left alignment, usually for text-derived classes.

HorizontalAlignment **HORIZONTAL_ALIGNMENT_CENTER** = `1`

Horizontal center alignment, usually for text-derived classes.

HorizontalAlignment **HORIZONTAL_ALIGNMENT_RIGHT** = `2`

Horizontal right alignment, usually for text-derived classes.

HorizontalAlignment **HORIZONTAL_ALIGNMENT_FILL** = `3`

Expand row to fit width, usually for text-derived classes.

---

enum **VerticalAlignment**: 🔗

VerticalAlignment **VERTICAL_ALIGNMENT_TOP** = `0`

Vertical top alignment, usually for text-derived classes.

VerticalAlignment **VERTICAL_ALIGNMENT_CENTER** = `1`

Vertical center alignment, usually for text-derived classes.

VerticalAlignment **VERTICAL_ALIGNMENT_BOTTOM** = `2`

Vertical bottom alignment, usually for text-derived classes.

VerticalAlignment **VERTICAL_ALIGNMENT_FILL** = `3`

Expand rows to fit height, usually for text-derived classes.

---

enum **InlineAlignment**: 🔗

InlineAlignment **INLINE_ALIGNMENT_TOP_TO** = `0`

Aligns the top of the inline object (e.g. image, table) to the position of the text specified by `INLINE_ALIGNMENT_TO_*` constant.

InlineAlignment **INLINE_ALIGNMENT_CENTER_TO** = `1`

Aligns the center of the inline object (e.g. image, table) to the position of the text specified by `INLINE_ALIGNMENT_TO_*` constant.

InlineAlignment **INLINE_ALIGNMENT_BASELINE_TO** = `3`

Aligns the baseline (user defined) of the inline object (e.g. image, table) to the position of the text specified by `INLINE_ALIGNMENT_TO_*` constant.

InlineAlignment **INLINE_ALIGNMENT_BOTTOM_TO** = `2`

Aligns the bottom of the inline object (e.g. image, table) to the position of the text specified by `INLINE_ALIGNMENT_TO_*` constant.

InlineAlignment **INLINE_ALIGNMENT_TO_TOP** = `0`

Aligns the position of the inline object (e.g. image, table) specified by `INLINE_ALIGNMENT_*_TO` constant to the top of the text.

InlineAlignment **INLINE_ALIGNMENT_TO_CENTER** = `4`

Aligns the position of the inline object (e.g. image, table) specified by `INLINE_ALIGNMENT_*_TO` constant to the center of the text.

InlineAlignment **INLINE_ALIGNMENT_TO_BASELINE** = `8`

Aligns the position of the inline object (e.g. image, table) specified by `INLINE_ALIGNMENT_*_TO` constant to the baseline of the text.

InlineAlignment **INLINE_ALIGNMENT_TO_BOTTOM** = `12`

Aligns inline object (e.g. image, table) to the bottom of the text.

InlineAlignment **INLINE_ALIGNMENT_TOP** = `0`

Aligns top of the inline object (e.g. image, table) to the top of the text. Equivalent to `INLINE_ALIGNMENT_TOP_TO | INLINE_ALIGNMENT_TO_TOP`.

InlineAlignment **INLINE_ALIGNMENT_CENTER** = `5`

Aligns center of the inline object (e.g. image, table) to the center of the text. Equivalent to `INLINE_ALIGNMENT_CENTER_TO | INLINE_ALIGNMENT_TO_CENTER`.

InlineAlignment **INLINE_ALIGNMENT_BOTTOM** = `14`

Aligns bottom of the inline object (e.g. image, table) to the bottom of the text. Equivalent to `INLINE_ALIGNMENT_BOTTOM_TO | INLINE_ALIGNMENT_TO_BOTTOM`.

InlineAlignment **INLINE_ALIGNMENT_IMAGE_MASK** = `3`

A bit mask for `INLINE_ALIGNMENT_*_TO` alignment constants.

InlineAlignment **INLINE_ALIGNMENT_TEXT_MASK** = `12`

A bit mask for `INLINE_ALIGNMENT_TO_*` alignment constants.

---

enum **EulerOrder**: 🔗

EulerOrder **EULER_ORDER_XYZ** = `0`

Specifies that Euler angles should be in XYZ order. When composing, the order is X, Y, Z. When decomposing, the order is reversed, first Z, then Y, and X last.

EulerOrder **EULER_ORDER_XZY** = `1`

Specifies that Euler angles should be in XZY order. When composing, the order is X, Z, Y. When decomposing, the order is reversed, first Y, then Z, and X last.

EulerOrder **EULER_ORDER_YXZ** = `2`

Specifies that Euler angles should be in YXZ order. When composing, the order is Y, X, Z. When decomposing, the order is reversed, first Z, then X, and Y last.

EulerOrder **EULER_ORDER_YZX** = `3`

Specifies that Euler angles should be in YZX order. When composing, the order is Y, Z, X. When decomposing, the order is reversed, first X, then Z, and Y last.

EulerOrder **EULER_ORDER_ZXY** = `4`

Specifies that Euler angles should be in ZXY order. When composing, the order is Z, X, Y. When decomposing, the order is reversed, first Y, then X, and Z last.

EulerOrder **EULER_ORDER_ZYX** = `5`

Specifies that Euler angles should be in ZYX order. When composing, the order is Z, Y, X. When decomposing, the order is reversed, first X, then Y, and Z last.

---

enum **Key**: 🔗

Key **KEY_NONE** = `0`

Enum value which doesn't correspond to any key. This is used to initialize Key properties with a generic state.

Key **KEY_SPECIAL** = `4194304`

Keycodes with this bit applied are non-printable.

Key **KEY_ESCAPE** = `4194305`

Escape key.

Key **KEY_TAB** = `4194306`

Tab key.

Key **KEY_BACKTAB** = `4194307`

Shift + Tab key.

Key **KEY_BACKSPACE** = `4194308`

Backspace key.

Key **KEY_ENTER** = `4194309`

Return key (on the main keyboard).

Key **KEY_KP_ENTER** = `4194310`

Enter key on the numeric keypad.

Key **KEY_INSERT** = `4194311`

Insert key.

Key **KEY_DELETE** = `4194312`

Delete key.

Key **KEY_PAUSE** = `4194313`

Pause key.

Key **KEY_PRINT** = `4194314`

Print Screen key.

Key **KEY_SYSREQ** = `4194315`

System Request key.

Key **KEY_CLEAR** = `4194316`

Clear key.

Key **KEY_HOME** = `4194317`

Home key.

Key **KEY_END** = `4194318`

End key.

Key **KEY_LEFT** = `4194319`

Left arrow key.

Key **KEY_UP** = `4194320`

Up arrow key.

Key **KEY_RIGHT** = `4194321`

Right arrow key.

Key **KEY_DOWN** = `4194322`

Down arrow key.

Key **KEY_PAGEUP** = `4194323`

Page Up key.

Key **KEY_PAGEDOWN** = `4194324`

Page Down key.

Key **KEY_SHIFT** = `4194325`

Shift key.

Key **KEY_CTRL** = `4194326`

Control key.

Key **KEY_META** = `4194327`

Meta key.

Key **KEY_ALT** = `4194328`

Alt key.

Key **KEY_CAPSLOCK** = `4194329`

Caps Lock key.

Key **KEY_NUMLOCK** = `4194330`

Num Lock key.

Key **KEY_SCROLLLOCK** = `4194331`

Scroll Lock key.

Key **KEY_F1** = `4194332`

F1 key.

Key **KEY_F2** = `4194333`

F2 key.

Key **KEY_F3** = `4194334`

F3 key.

Key **KEY_F4** = `4194335`

F4 key.

Key **KEY_F5** = `4194336`

F5 key.

Key **KEY_F6** = `4194337`

F6 key.

Key **KEY_F7** = `4194338`

F7 key.

Key **KEY_F8** = `4194339`

F8 key.

Key **KEY_F9** = `4194340`

F9 key.

Key **KEY_F10** = `4194341`

F10 key.

Key **KEY_F11** = `4194342`

F11 key.

Key **KEY_F12** = `4194343`

F12 key.

Key **KEY_F13** = `4194344`

F13 key.

Key **KEY_F14** = `4194345`

F14 key.

Key **KEY_F15** = `4194346`

F15 key.

Key **KEY_F16** = `4194347`

F16 key.

Key **KEY_F17** = `4194348`

F17 key.

Key **KEY_F18** = `4194349`

F18 key.

Key **KEY_F19** = `4194350`

F19 key.

Key **KEY_F20** = `4194351`

F20 key.

Key **KEY_F21** = `4194352`

F21 key.

Key **KEY_F22** = `4194353`

F22 key.

Key **KEY_F23** = `4194354`

F23 key.

Key **KEY_F24** = `4194355`

F24 key.

Key **KEY_F25** = `4194356`

F25 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_F26** = `4194357`

F26 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_F27** = `4194358`

F27 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_F28** = `4194359`

F28 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_F29** = `4194360`

F29 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_F30** = `4194361`

F30 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_F31** = `4194362`

F31 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_F32** = `4194363`

F32 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_F33** = `4194364`

F33 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_F34** = `4194365`

F34 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_F35** = `4194366`

F35 key. Only supported on macOS and Linux due to a Windows limitation.

Key **KEY_KP_MULTIPLY** = `4194433`

Multiply (*) key on the numeric keypad.

Key **KEY_KP_DIVIDE** = `4194434`

Divide (/) key on the numeric keypad.

Key **KEY_KP_SUBTRACT** = `4194435`

Subtract (-) key on the numeric keypad.

Key **KEY_KP_PERIOD** = `4194436`

Period (.) key on the numeric keypad.

Key **KEY_KP_ADD** = `4194437`

Add (+) key on the numeric keypad.

Key **KEY_KP_0** = `4194438`

Number 0 on the numeric keypad.

Key **KEY_KP_1** = `4194439`

Number 1 on the numeric keypad.

Key **KEY_KP_2** = `4194440`

Number 2 on the numeric keypad.

Key **KEY_KP_3** = `4194441`

Number 3 on the numeric keypad.

Key **KEY_KP_4** = `4194442`

Number 4 on the numeric keypad.

Key **KEY_KP_5** = `4194443`

Number 5 on the numeric keypad.

Key **KEY_KP_6** = `4194444`

Number 6 on the numeric keypad.

Key **KEY_KP_7** = `4194445`

Number 7 on the numeric keypad.

Key **KEY_KP_8** = `4194446`

Number 8 on the numeric keypad.

Key **KEY_KP_9** = `4194447`

Number 9 on the numeric keypad.

Key **KEY_MENU** = `4194370`

Context menu key.

Key **KEY_HYPER** = `4194371`

Hyper key. (On Linux/X11 only).

Key **KEY_HELP** = `4194373`

Help key.

Key **KEY_BACK** = `4194376`

Media back key. Not to be confused with the Back button on an Android device.

Key **KEY_FORWARD** = `4194377`

Media forward key.

Key **KEY_STOP** = `4194378`

Media stop key.

Key **KEY_REFRESH** = `4194379`

Media refresh key.

Key **KEY_VOLUMEDOWN** = `4194380`

Volume down key.

Key **KEY_VOLUMEMUTE** = `4194381`

Mute volume key.

Key **KEY_VOLUMEUP** = `4194382`

Volume up key.

Key **KEY_MEDIAPLAY** = `4194388`

Media play key.

Key **KEY_MEDIASTOP** = `4194389`

Media stop key.

Key **KEY_MEDIAPREVIOUS** = `4194390`

Previous song key.

Key **KEY_MEDIANEXT** = `4194391`

Next song key.

Key **KEY_MEDIARECORD** = `4194392`

Media record key.

Key **KEY_HOMEPAGE** = `4194393`

Home page key.

Key **KEY_FAVORITES** = `4194394`

Favorites key.

Key **KEY_SEARCH** = `4194395`

Search key.

Key **KEY_STANDBY** = `4194396`

Standby key.

Key **KEY_OPENURL** = `4194397`

Open URL / Launch Browser key.

Key **KEY_LAUNCHMAIL** = `4194398`

Launch Mail key.

Key **KEY_LAUNCHMEDIA** = `4194399`

Launch Media key.

Key **KEY_LAUNCH0** = `4194400`

Launch Shortcut 0 key.

Key **KEY_LAUNCH1** = `4194401`

Launch Shortcut 1 key.

Key **KEY_LAUNCH2** = `4194402`

Launch Shortcut 2 key.

Key **KEY_LAUNCH3** = `4194403`

Launch Shortcut 3 key.

Key **KEY_LAUNCH4** = `4194404`

Launch Shortcut 4 key.

Key **KEY_LAUNCH5** = `4194405`

Launch Shortcut 5 key.

Key **KEY_LAUNCH6** = `4194406`

Launch Shortcut 6 key.

Key **KEY_LAUNCH7** = `4194407`

Launch Shortcut 7 key.

Key **KEY_LAUNCH8** = `4194408`

Launch Shortcut 8 key.

Key **KEY_LAUNCH9** = `4194409`

Launch Shortcut 9 key.

Key **KEY_LAUNCHA** = `4194410`

Launch Shortcut A key.

Key **KEY_LAUNCHB** = `4194411`

Launch Shortcut B key.

Key **KEY_LAUNCHC** = `4194412`

Launch Shortcut C key.

Key **KEY_LAUNCHD** = `4194413`

Launch Shortcut D key.

Key **KEY_LAUNCHE** = `4194414`

Launch Shortcut E key.

Key **KEY_LAUNCHF** = `4194415`

Launch Shortcut F key.

Key **KEY_GLOBE** = `4194416`

"Globe" key on Mac / iPad keyboard.

Key **KEY_KEYBOARD** = `4194417`

"On-screen keyboard" key on iPad keyboard.

Key **KEY_JIS_EISU** = `4194418`

英数 key on Mac keyboard.

Key **KEY_JIS_KANA** = `4194419`

かな key on Mac keyboard.

Key **KEY_UNKNOWN** = `8388607`

Unknown key.

Key **KEY_SPACE** = `32`

Space key.

Key **KEY_EXCLAM** = `33`

! key.

Key **KEY_QUOTEDBL** = `34`

" key.

Key **KEY_NUMBERSIGN** = `35`

# key.

Key **KEY_DOLLAR** = `36`

$ key.

Key **KEY_PERCENT** = `37`

% key.

Key **KEY_AMPERSAND** = `38`

& key.

Key **KEY_APOSTROPHE** = `39`

' key.

Key **KEY_PARENLEFT** = `40`

(key.

Key **KEY_PARENRIGHT** = `41`

) key.

Key **KEY_ASTERISK** = `42`

* key.

Key **KEY_PLUS** = `43`

- key.

Key **KEY_COMMA** = `44`

, key.

Key **KEY_MINUS** = `45`

- key.

Key **KEY_PERIOD** = `46`

. key.

Key **KEY_SLASH** = `47`

/ key.

Key **KEY_0** = `48`

Number 0 key.

Key **KEY_1** = `49`

Number 1 key.

Key **KEY_2** = `50`

Number 2 key.

Key **KEY_3** = `51`

Number 3 key.

Key **KEY_4** = `52`

Number 4 key.

Key **KEY_5** = `53`

Number 5 key.

Key **KEY_6** = `54`

Number 6 key.

Key **KEY_7** = `55`

Number 7 key.

Key **KEY_8** = `56`

Number 8 key.

Key **KEY_9** = `57`

Number 9 key.

Key **KEY_COLON** = `58`

: key.

Key **KEY_SEMICOLON** = `59`

; key.

Key **KEY_LESS** = `60`

< key.

Key **KEY_EQUAL** = `61`

= key.

Key **KEY_GREATER** = `62`

> key.

Key **KEY_QUESTION** = `63`

? key.

Key **KEY_AT** = `64`

@ key.

Key **KEY_A** = `65`

A key.

Key **KEY_B** = `66`

B key.

Key **KEY_C** = `67`

C key.

Key **KEY_D** = `68`

D key.

Key **KEY_E** = `69`

E key.

Key **KEY_F** = `70`

F key.

Key **KEY_G** = `71`

G key.

Key **KEY_H** = `72`

H key.

Key **KEY_I** = `73`

I key.

Key **KEY_J** = `74`

J key.

Key **KEY_K** = `75`

K key.

Key **KEY_L** = `76`

L key.

Key **KEY_M** = `77`

M key.

Key **KEY_N** = `78`

N key.

Key **KEY_O** = `79`

O key.

Key **KEY_P** = `80`

P key.

Key **KEY_Q** = `81`

Q key.

Key **KEY_R** = `82`

R key.

Key **KEY_S** = `83`

S key.

Key **KEY_T** = `84`

T key.

Key **KEY_U** = `85`

U key.

Key **KEY_V** = `86`

V key.

Key **KEY_W** = `87`

W key.

Key **KEY_X** = `88`

X key.

Key **KEY_Y** = `89`

Y key.

Key **KEY_Z** = `90`

Z key.

Key **KEY_BRACKETLEFT** = `91`

[key.

Key **KEY_BACKSLASH** = `92`

\ key.

Key **KEY_BRACKETRIGHT** = `93`

] key.

Key **KEY_ASCIICIRCUM** = `94`

^ key.

Key **KEY_UNDERSCORE** = `95`

_ key.

Key **KEY_QUOTELEFT** = `96`

` key.

Key **KEY_BRACELEFT** = `123`

{ key.

Key **KEY_BAR** = `124`

key.

Key **KEY_BRACERIGHT** = `125`

} key.

Key **KEY_ASCIITILDE** = `126`

~ key.

Key **KEY_YEN** = `165`

¥ key.

Key **KEY_SECTION** = `167`

§ key.

---

flags **KeyModifierMask**: 🔗

KeyModifierMask **KEY_CODE_MASK** = `8388607`

Key Code mask.

KeyModifierMask **KEY_MODIFIER_MASK** = `532676608`

Modifier key mask.

KeyModifierMask **KEY_MASK_CMD_OR_CTRL** = `16777216`

Automatically remapped to KEY_META on macOS and KEY_CTRL on other platforms, this mask is never set in the actual events, and should be used for key mapping only.

KeyModifierMask **KEY_MASK_SHIFT** = `33554432`

Shift key mask.

KeyModifierMask **KEY_MASK_ALT** = `67108864`

Alt or Option (on macOS) key mask.

KeyModifierMask **KEY_MASK_META** = `134217728`

Command (on macOS) or Meta/Windows key mask.

KeyModifierMask **KEY_MASK_CTRL** = `268435456`

Control key mask.

KeyModifierMask **KEY_MASK_KPAD** = `536870912`

Keypad key mask.

KeyModifierMask **KEY_MASK_GROUP_SWITCH** = `1073741824`

Group Switch key mask.

---

enum **KeyLocation**: 🔗

KeyLocation **KEY_LOCATION_UNSPECIFIED** = `0`

Used for keys which only appear once, or when a comparison doesn't need to differentiate the `LEFT` and `RIGHT` versions.

For example, when using InputEvent.is_match, an event which has KEY_LOCATION_UNSPECIFIED will match any KeyLocation on the passed event.

KeyLocation **KEY_LOCATION_LEFT** = `1`

A key which is to the left of its twin.

KeyLocation **KEY_LOCATION_RIGHT** = `2`

A key which is to the right of its twin.

enum **MouseButton**: 🔗

MouseButton **MOUSE_BUTTON_NONE** = `0`

Enum value which doesn't correspond to any mouse button. This is used to initialize MouseButton properties with a generic state.

MouseButton **MOUSE_BUTTON_LEFT** = `1`

Primary mouse button, usually assigned to the left button.

MouseButton **MOUSE_BUTTON_RIGHT** = `2`

Secondary mouse button, usually assigned to the right button.

MouseButton **MOUSE_BUTTON_MIDDLE** = `3`

Middle mouse button.

MouseButton **MOUSE_BUTTON_WHEEL_UP** = `4`

Mouse wheel scrolling up.

MouseButton **MOUSE_BUTTON_WHEEL_DOWN** = `5`

Mouse wheel scrolling down.

MouseButton **MOUSE_BUTTON_WHEEL_LEFT** = `6`

Mouse wheel left button (only present on some mice).

MouseButton **MOUSE_BUTTON_WHEEL_RIGHT** = `7`

Mouse wheel right button (only present on some mice).

MouseButton **MOUSE_BUTTON_XBUTTON1** = `8`

Extra mouse button 1. This is sometimes present, usually to the sides of the mouse.

MouseButton **MOUSE_BUTTON_XBUTTON2** = `9`

Extra mouse button 2. This is sometimes present, usually to the sides of the mouse.

flags **MouseButtonMask**: 🔗

MouseButtonMask **MOUSE_BUTTON_MASK_LEFT** = `1`

Primary mouse button mask, usually for the left button.

MouseButtonMask **MOUSE_BUTTON_MASK_RIGHT** = `2`

Secondary mouse button mask, usually for the right button.

MouseButtonMask **MOUSE_BUTTON_MASK_MIDDLE** = `4`

Middle mouse button mask.

MouseButtonMask **MOUSE_BUTTON_MASK_MB_XBUTTON1** = `128`

Extra mouse button 1 mask.

MouseButtonMask **MOUSE_BUTTON_MASK_MB_XBUTTON2** = `256`

Extra mouse button 2 mask.

enum **JoyButton**: 🔗

JoyButton **JOY_BUTTON_INVALID** = `-1`

An invalid game controller button.

JoyButton **JOY_BUTTON_A** = `0`

Game controller SDL button A. Corresponds to the bottom action button: Sony Cross, Xbox A, Nintendo B.

JoyButton **JOY_BUTTON_B** = `1`

Game controller SDL button B. Corresponds to the right action button: Sony Circle, Xbox B, Nintendo A.

JoyButton **JOY_BUTTON_X** = `2`

Game controller SDL button X. Corresponds to the left action button: Sony Square, Xbox X, Nintendo Y.

JoyButton **JOY_BUTTON_Y** = `3`

Game controller SDL button Y. Corresponds to the top action button: Sony Triangle, Xbox Y, Nintendo X.

JoyButton **JOY_BUTTON_BACK** = `4`

Game controller SDL back button. Corresponds to the Sony Select, Xbox Back, Nintendo - button.

JoyButton **JOY_BUTTON_GUIDE** = `5`

Game controller SDL guide button. Corresponds to the Sony PS, Xbox Home button.

JoyButton **JOY_BUTTON_START** = `6`

Game controller SDL start button. Corresponds to the Sony Options, Xbox Menu, Nintendo + button.

JoyButton **JOY_BUTTON_LEFT_STICK** = `7`

Game controller SDL left stick button. Corresponds to the Sony L3, Xbox L/LS button.

JoyButton **JOY_BUTTON_RIGHT_STICK** = `8`

Game controller SDL right stick button. Corresponds to the Sony R3, Xbox R/RS button.

JoyButton **JOY_BUTTON_LEFT_SHOULDER** = `9`

Game controller SDL left shoulder button. Corresponds to the Sony L1, Xbox LB button.

JoyButton **JOY_BUTTON_RIGHT_SHOULDER** = `10`

Game controller SDL right shoulder button. Corresponds to the Sony R1, Xbox RB button.

JoyButton **JOY_BUTTON_DPAD_UP** = `11`

Game controller D-pad up button.

JoyButton **JOY_BUTTON_DPAD_DOWN** = `12`

Game controller D-pad down button.

JoyButton **JOY_BUTTON_DPAD_LEFT** = `13`

Game controller D-pad left button.

JoyButton **JOY_BUTTON_DPAD_RIGHT** = `14`

Game controller D-pad right button.

JoyButton **JOY_BUTTON_MISC1** = `15`

Game controller SDL miscellaneous button. Corresponds to Xbox share button, PS5 microphone button, Nintendo Switch capture button.

JoyButton **JOY_BUTTON_PADDLE1** = `16`

Game controller SDL paddle 1 button.

JoyButton **JOY_BUTTON_PADDLE2** = `17`

Game controller SDL paddle 2 button.

JoyButton **JOY_BUTTON_PADDLE3** = `18`

Game controller SDL paddle 3 button.

JoyButton **JOY_BUTTON_PADDLE4** = `19`

Game controller SDL paddle 4 button.

JoyButton **JOY_BUTTON_TOUCHPAD** = `20`

Game controller SDL touchpad button.

JoyButton **JOY_BUTTON_SDL_MAX** = `21`

The number of SDL game controller buttons.

JoyButton **JOY_BUTTON_MAX** = `128`

The maximum number of game controller buttons supported by the engine. The actual limit may be lower on specific platforms:

- **Android:** Up to 36 buttons.
- **Linux:** Up to 80 buttons.
- **Windows** and **macOS:** Up to 128 buttons.

enum **JoyAxis**: 🔗

JoyAxis **JOY_AXIS_INVALID** = `-1`

An invalid game controller axis.

JoyAxis **JOY_AXIS_LEFT_X** = `0`

Game controller left joystick x-axis.

JoyAxis **JOY_AXIS_LEFT_Y** = `1`

Game controller left joystick y-axis.

JoyAxis **JOY_AXIS_RIGHT_X** = `2`

Game controller right joystick x-axis.

JoyAxis **JOY_AXIS_RIGHT_Y** = `3`

Game controller right joystick y-axis.

JoyAxis **JOY_AXIS_TRIGGER_LEFT** = `4`

Game controller left trigger axis.

JoyAxis **JOY_AXIS_TRIGGER_RIGHT** = `5`

Game controller right trigger axis.

JoyAxis **JOY_AXIS_SDL_MAX** = `6`

The number of SDL game controller axes.

JoyAxis **JOY_AXIS_MAX** = `10`

The maximum number of game controller axes: OpenVR supports up to 5 Joysticks making a total of 10 axes.

---

enum **MIDIMessage**: 🔗

MIDIMessage **MIDI_MESSAGE_NONE** = `0`

Does not correspond to any MIDI message. This is the default value of InputEventMIDI.message.

MIDIMessage **MIDI_MESSAGE_NOTE_OFF** = `8`

MIDI message sent when a note is released.

Note: Not all MIDI devices send this message; some may send MIDI_MESSAGE_NOTE_ON with InputEventMIDI.velocity set to `0`.

MIDIMessage **MIDI_MESSAGE_NOTE_ON** = `9`

MIDI message sent when a note is pressed.

MIDIMessage **MIDI_MESSAGE_AFTERTOUCH** = `10`

MIDI message sent to indicate a change in pressure while a note is being pressed down, also called aftertouch.

MIDIMessage **MIDI_MESSAGE_CONTROL_CHANGE** = `11`

MIDI message sent when a controller value changes. In a MIDI device, a controller is any input that doesn't play notes. These may include sliders for volume, balance, and panning, as well as switches and pedals. See the General MIDI specification [https://en.wikipedia.org/wiki/General_MIDI#Controller_events] for a small list.

MIDIMessage **MIDI_MESSAGE_PROGRAM_CHANGE** = `12`

MIDI message sent when the MIDI device changes its current instrument (also called *program* or *preset*).

MIDIMessage **MIDI_MESSAGE_CHANNEL_PRESSURE** = `13`

MIDI message sent to indicate a change in pressure for the whole channel. Some MIDI devices may send this instead of MIDI_MESSAGE_AFTERTOUCH.

MIDIMessage **MIDI_MESSAGE_PITCH_BEND** = `14`

MIDI message sent when the value of the pitch bender changes, usually a wheel on the MIDI device.

MIDIMessage **MIDI_MESSAGE_SYSTEM_EXCLUSIVE** = `240`

MIDI system exclusive (SysEx) message. This type of message is not standardized and it's highly dependent on the MIDI device sending it.

Note: Getting this message's data from InputEventMIDI is not implemented.

MIDIMessage **MIDI_MESSAGE_QUARTER_FRAME** = `241`

MIDI message sent every quarter frame to keep connected MIDI devices synchronized. Related to MIDI_MESSAGE_TIMING_CLOCK.

Note: Getting this message's data from InputEventMIDI is not implemented.

MIDIMessage **MIDI_MESSAGE_SONG_POSITION_POINTER** = `242`

MIDI message sent to jump onto a new position in the current sequence or song.

Note: Getting this message's data from InputEventMIDI is not implemented.

MIDIMessage **MIDI_MESSAGE_SONG_SELECT** = `243`

MIDI message sent to select a sequence or song to play.

Note: Getting this message's data from InputEventMIDI is not implemented.

MIDIMessage **MIDI_MESSAGE_TUNE_REQUEST** = `246`

MIDI message sent to request a tuning calibration. Used on analog synthesizers. Most modern MIDI devices do not need this message.

MIDIMessage **MIDI_MESSAGE_TIMING_CLOCK** = `248`

MIDI message sent 24 times after MIDI_MESSAGE_QUARTER_FRAME, to keep connected MIDI devices synchronized.

MIDIMessage **MIDI_MESSAGE_START** = `250`

MIDI message sent to start the current sequence or song from the beginning.

MIDIMessage **MIDI_MESSAGE_CONTINUE** = `251`

MIDI message sent to resume from the point the current sequence or song was paused.

MIDIMessage **MIDI_MESSAGE_STOP** = `252`

MIDI message sent to pause the current sequence or song.

MIDIMessage **MIDI_MESSAGE_ACTIVE_SENSING** = `254`

MIDI message sent repeatedly while the MIDI device is idle, to tell the receiver that the connection is alive. Most MIDI devices do not send this message.

MIDIMessage **MIDI_MESSAGE_SYSTEM_RESET** = `255`

MIDI message sent to reset a MIDI device to its default state, as if it was just turned on. It should not be sent when the MIDI device is being turned on.

---

enum **Error**: 🔗

Error **OK** = `0`

Methods that return Error return OK when no error occurred.

Since OK has value 0, and all other error constants are positive integers, it can also be used in boolean checks.

Example:

```
var error = method_that_returns_error()
if error != OK:
    printerr("Failure!")

# Or, alternatively:
if error:
    printerr("Still failing!")
```

Note: Many functions do not return an error code, but will print error messages to standard output.

Error **FAILED** = `1`

Generic error.

Error **ERR_UNAVAILABLE** = `2`

Unavailable error.

Error **ERR_UNCONFIGURED** = `3`

Unconfigured error.

Error **ERR_UNAUTHORIZED** = `4`

Unauthorized error.

Error **ERR_PARAMETER_RANGE_ERROR** = `5`

Parameter range error.

Error **ERR_OUT_OF_MEMORY** = `6`

Out of memory (OOM) error.

Error **ERR_FILE_NOT_FOUND** = `7`

File: Not found error.

Error **ERR_FILE_BAD_DRIVE** = `8`

File: Bad drive error.

Error **ERR_FILE_BAD_PATH** = `9`

File: Bad path error.

Error **ERR_FILE_NO_PERMISSION** = `10`

File: No permission error.

Error **ERR_FILE_ALREADY_IN_USE** = `11`

File: Already in use error.

Error **ERR_FILE_CANT_OPEN** = `12`

File: Can't open error.

Error **ERR_FILE_CANT_WRITE** = `13`

File: Can't write error.

Error **ERR_FILE_CANT_READ** = `14`

File: Can't read error.

Error **ERR_FILE_UNRECOGNIZED** = `15`

File: Unrecognized error.

Error **ERR_FILE_CORRUPT** = `16`

File: Corrupt error.

Error **ERR_FILE_MISSING_DEPENDENCIES** = `17`

File: Missing dependencies error.

Error **ERR_FILE_EOF** = `18`

File: End of file (EOF) error.

Error **ERR_CANT_OPEN** = `19`

Can't open error.

Error **ERR_CANT_CREATE** = `20`

Can't create error.

Error **ERR_QUERY_FAILED** = `21`

Query failed error.

Error **ERR_ALREADY_IN_USE** = `22`

Already in use error.

Error **ERR_LOCKED** = `23`

Locked error.

Error **ERR_TIMEOUT** = `24`

Timeout error.

Error **ERR_CANT_CONNECT** = `25`

Can't connect error.

Error **ERR_CANT_RESOLVE** = `26`

Can't resolve error.

Error **ERR_CONNECTION_ERROR** = `27`

Connection error.

Error **ERR_CANT_ACQUIRE_RESOURCE** = `28`

Can't acquire resource error.

Error **ERR_CANT_FORK** = `29`

Can't fork process error.

Error **ERR_INVALID_DATA** = `30`

Invalid data error.

Error **ERR_INVALID_PARAMETER** = `31`

Invalid parameter error.

Error **ERR_ALREADY_EXISTS** = `32`

Already exists error.

Error **ERR_DOES_NOT_EXIST** = `33`

Does not exist error.

Error **ERR_DATABASE_CANT_READ** = `34`

Database: Read error.

Error **ERR_DATABASE_CANT_WRITE** = `35`

Database: Write error.

Error **ERR_COMPILATION_FAILED** = `36`

Compilation failed error.

Error **ERR_METHOD_NOT_FOUND** = `37`

Method not found error.

Error **ERR_LINK_FAILED** = `38`

Linking failed error.

Error **ERR_SCRIPT_FAILED** = `39`

Script failed error.

Error **ERR_CYCLIC_LINK** = `40`

Cycling link (import cycle) error.

Error **ERR_INVALID_DECLARATION** = `41`

Invalid declaration error.

Error **ERR_DUPLICATE_SYMBOL** = `42`

Duplicate symbol error.

Error **ERR_PARSE_ERROR** = `43`

Parse error.

Error **ERR_BUSY** = `44`

Busy error.

Error **ERR_SKIP** = `45`

Skip error.

Error **ERR_HELP** = `46`

Help error. Used internally when passing `--version` or `--help` as executable options.

Error **ERR_BUG** = `47`

Bug error, caused by an implementation issue in the method.

Note: If a built-in method returns this code, please open an issue on the GitHub Issue Tracker [https://github.com/godotengine/godot/issues].

Error **ERR_PRINTER_ON_FIRE** = `48`

Printer on fire error (This is an easter egg, no built-in methods return this error code).

---

enum **PropertyHint**: 🔗

PropertyHint **PROPERTY_HINT_NONE** = `0`

The property has no hint for the editor.

PropertyHint **PROPERTY_HINT_RANGE** = `1`

Hints that an int or float property should be within a range specified via the hint string `"min,max"` or `"min,max,step"`. The hint string can optionally include `"or_greater"` and/or `"or_less"` to allow manual input going respectively above the max or below the min values.

Example: `"-360,360,1,or_greater,or_less"`.

Additionally, other keywords can be included: `"exp"` for exponential range editing, `"radians_as_degrees"` for editing radian angles in degrees (the range values are also in degrees), `"degrees"` to hint at an angle and `"hide_slider"` to hide the slider.

PropertyHint **PROPERTY_HINT_ENUM** = `2`

Hints that an int or String property is an enumerated value to pick in a list specified via a hint string.

The hint string is a comma separated list of names such as `"Hello,Something,Else"`. Whitespaces are **not** removed from either end of a name. For integer properties, the first name in the list has value 0, the next 1, and so on. Explicit values can also be specified by appending `:integer` to the name, e.g. `"Zero,One,Three:3,Four,Six:6"`.

PropertyHint **PROPERTY_HINT_ENUM_SUGGESTION** = `3`

Hints that a String property can be an enumerated value to pick in a list specified via a hint string such as `"Hello,Something,Else"`.

Unlike PROPERTY_HINT_ENUM, a property with this hint still accepts arbitrary values and can be empty. The list of values serves to suggest possible values.

PropertyHint **PROPERTY_HINT_EXP_EASING** = `4`

Hints that a float property should be edited via an exponential easing function. The hint string can include `"attenuation"` to flip the curve horizontally and/or `"positive_only"` to exclude in/out easing and limit values to be greater than or equal to zero.

PropertyHint **PROPERTY_HINT_LINK** = `5`

Hints that a vector property should allow its components to be linked. For example, this allows Vector2.x and Vector2.y to be edited together.

PropertyHint **PROPERTY_HINT_FLAGS** = `6`

Hints that an int property is a bitmask with named bit flags.

The hint string is a comma separated list of names such as `"Bit0,Bit1,Bit2,Bit3"`. Whitespaces are **not** removed from either end of a name. The first name in the list has value 1, the next 2, then 4, 8, 16 and so on. Explicit values can also be specified by appending `:integer` to the name, e.g. `"A:4,B:8,C:16"`. You can also combine several flags (`"A:4,B:8,AB:12,C:16"`).

Note: A flag value must be at least `1` and at most `2 ** 32 - 1`.

Note: Unlike PROPERTY_HINT_ENUM, the previous explicit value is not taken into account. For the hint `"A:16,B,C"`, A is 16, B is 2, C is 4.

PropertyHint **PROPERTY_HINT_LAYERS_2D_RENDER** = `7`

Hints that an int property is a bitmask using the optionally named 2D render layers.

PropertyHint **PROPERTY_HINT_LAYERS_2D_PHYSICS** = `8`

Hints that an int property is a bitmask using the optionally named 2D physics layers.

PropertyHint **PROPERTY_HINT_LAYERS_2D_NAVIGATION** = `9`

Hints that an int property is a bitmask using the optionally named 2D navigation layers.

PropertyHint **PROPERTY_HINT_LAYERS_3D_RENDER** = `10`

Hints that an int property is a bitmask using the optionally named 3D render layers.

PropertyHint **PROPERTY_HINT_LAYERS_3D_PHYSICS** = `11`

Hints that an int property is a bitmask using the optionally named 3D physics layers.

PropertyHint **PROPERTY_HINT_LAYERS_3D_NAVIGATION** = `12`

Hints that an int property is a bitmask using the optionally named 3D navigation layers.

PropertyHint **PROPERTY_HINT_LAYERS_AVOIDANCE** = `37`

Hints that an integer property is a bitmask using the optionally named avoidance layers.

PropertyHint **PROPERTY_HINT_FILE** = `13`

Hints that a String property is a path to a file. Editing it will show a file dialog for picking the path. The hint string can be a set of filters with wildcards like `"*.png,*.jpg"`.

PropertyHint **PROPERTY_HINT_DIR** = `14`

Hints that a String property is a path to a directory. Editing it will show a file dialog for picking the path.

PropertyHint **PROPERTY_HINT_GLOBAL_FILE** = `15`

Hints that a String property is an absolute path to a file outside the project folder. Editing it will show a file dialog for picking the path. The hint string can be a set of filters with wildcards, like `"*.png,*.jpg"`.

PropertyHint **PROPERTY_HINT_GLOBAL_DIR** = `16`

Hints that a String property is an absolute path to a directory outside the project folder. Editing it will show a file dialog for picking the path.

PropertyHint **PROPERTY_HINT_RESOURCE_TYPE** = `17`

Hints that a property is an instance of a Resource-derived type, optionally specified via the hint string (e.g. `"Texture2D"`). Editing it will show a popup menu of valid resource types to instantiate.

PropertyHint **PROPERTY_HINT_MULTILINE_TEXT** = `18`

Hints that a String property is text with line breaks. Editing it will show a text input field where line breaks can be typed.

PropertyHint **PROPERTY_HINT_EXPRESSION** = `19`

Hints that a String property is an Expression.

PropertyHint **PROPERTY_HINT_PLACEHOLDER_TEXT** = `20`

Hints that a String property should show a placeholder text on its input field, if empty. The hint string is the placeholder text to use.

PropertyHint **PROPERTY_HINT_COLOR_NO_ALPHA** = `21`

Hints that a Color property should be edited without affecting its transparency (Color.a is not editable).

PropertyHint **PROPERTY_HINT_OBJECT_ID** = `22`

Hints that the property's value is an object encoded as object ID, with its type specified in the hint string. Used by the debugger.

PropertyHint **PROPERTY_HINT_TYPE_STRING** = `23`

If a property is String, hints that the property represents a particular type (class). This allows to select a type from the create dialog. The property will store the selected type as a string.

If a property is Array, hints the editor how to show elements. The `hint_string` must encode nested types using `":"` and `"/"`.

GDScript

```
# Array of elem_type.
hint_string = "%d:" % [elem_type]
hint_string = "%d/%d:%s" % [elem_type, elem_hint,
elem_hint_string]
# Two-dimensional array of elem_type (array of arrays of
elem_type).
hint_string = "%d:%d:" % [TYPE_ARRAY, elem_type]
hint_string = "%d:%d/%d:%s" % [TYPE_ARRAY, elem_type,
elem_hint, elem_hint_string]
# Three-dimensional array of elem_type (array of arrays of
arrays of elem_type).
hint_string = "%d:%d:%d:" % [TYPE_ARRAY, TYPE_ARRAY, elem_type]
hint_string = "%d:%d:%d/%d:%s" % [TYPE_ARRAY, TYPE_ARRAY,
elem_type, elem_hint, elem_hint_string]
```

C#

```
// Array of elemType.
hintString = $"{elemType:D}:";
hintString = $"{elemType:}/{elemHint:D}:{elemHintString}";
// Two-dimensional array of elemType (array of arrays of
elemType).
hintString = $"{Variant.Type.Array:D}:{elemType:D}:";
hintString = $"{Variant.Type.Array:D}:
{elemType:D}/{elemHint:D}:{elemHintString}";
// Three-dimensional array of elemType (array of arrays of
arrays of elemType).
hintString = $"{Variant.Type.Array:D}:{Variant.Type.Array:D}:
{elemType:D}:";
hintString = $"{Variant.Type.Array:D}:{Variant.Type.Array:D}:
{elemType:D}/{elemHint:D}:{elemHintString}";
```

Examples:

GDScript

```
hint_string = "%d:" % [TYPE_INT] # Array of integers.
hint_string = "%d/%d:1,10,1" % [TYPE_INT, PROPERTY_HINT_RANGE]
# Array of integers (in range from 1 to 10).
hint_string = "%d/%d:Zero,One,Two" % [TYPE_INT,
PROPERTY_HINT_ENUM] # Array of integers (an enum).
hint_string = "%d/%d:Zero,One,Three:3,Six:6" % [TYPE_INT,
PROPERTY_HINT_ENUM] # Array of integers (an enum).
hint_string = "%d/%d:*.png" % [TYPE_STRING, PROPERTY_HINT_FILE]
# Array of strings (file paths).
hint_string = "%d/%d:Texture2D" % [TYPE_OBJECT,
PROPERTY_HINT_RESOURCE_TYPE] # Array of textures.

hint_string = "%d:%d:" % [TYPE_ARRAY, TYPE_FLOAT] # Two-
dimensional array of floats.
hint_string = "%d:%d/%d:" % [TYPE_ARRAY, TYPE_STRING,
PROPERTY_HINT_MULTILINE_TEXT] # Two-dimensional array of
multiline strings.
hint_string = "%d:%d/%d:-1,1,0.1" % [TYPE_ARRAY, TYPE_FLOAT,
PROPERTY_HINT_RANGE] # Two-dimensional array of floats (in
range from -1 to 1).
hint_string = "%d:%d/%d:Texture2D" % [TYPE_ARRAY, TYPE_OBJECT,
PROPERTY_HINT_RESOURCE_TYPE] # Two-dimensional array of
textures.
```

C#

```
hintString = $"{Variant.Type.Int:D}/{PropertyHint.Range:D}:1,10,1"; // Array of integers (in range from 1 to 10).
hintString = $"{Variant.Type.Int:D}/{PropertyHint.Enum:D}:Zero,One,Two"; // Array of integers (an enum).
hintString = $"{Variant.Type.Int:D}/{PropertyHint.Enum:D}:Zero,One,Three:3,Six:6"; // Array of integers (an enum).
hintString = $"{Variant.Type.String:D}/{PropertyHint.File:D}:*.png"; // Array of strings (file paths).
hintString = $"{Variant.Type.Object:D}/{PropertyHint.ResourceType:D}:Texture2D"; // Array of textures.

hintString = $"{Variant.Type.Array:D}:{Variant.Type.Float:D}:"; // Two-dimensional array of floats.
hintString = $"{Variant.Type.Array:D}:{Variant.Type.String:D}/{PropertyHint.MultilineText:D}:"; // Two-dimensional array of multiline strings.
hintString = $"{Variant.Type.Array:D}:{Variant.Type.Float:D}/{PropertyHint.Range:D}:-1,1,0.1"; // Two-dimensional array of floats (in range from -1 to 1).
hintString = $"{Variant.Type.Array:D}:{Variant.Type.Object:D}/{PropertyHint.ResourceType:D}:Texture2D"; // Two-dimensional array of textures.
```

Note: The trailing colon is required for properly detecting built-in types.

PropertyHint **PROPERTY_HINT_NODE_PATH_TO_EDITED_NODE** = `24`

Deprecated: This hint is not used by the engine.

PropertyHint **PROPERTY_HINT_OBJECT_TOO_BIG** = `25`

Hints that an object is too big to be sent via the debugger.

PropertyHint **PROPERTY_HINT_NODE_PATH_VALID_TYPES** = `26`

Hints that the hint string specifies valid node types for property of type NodePath.

PropertyHint **PROPERTY_HINT_SAVE_FILE** = `27`

Hints that a String property is a path to a file. Editing it will show a file dialog for picking the path for the file to be saved at. The dialog has access to the project's directory. The hint string can be a set of filters with wildcards like `"*.png,*.jpg"`. See also FileDialog.filters.

PropertyHint **PROPERTY_HINT_GLOBAL_SAVE_FILE** = `28`

Hints that a String property is a path to a file. Editing it will show a file dialog for picking the path for the file to be saved at. The dialog has access to the entire filesystem. The hint string can be a set of filters with wildcards like `"*.png,*.jpg"`. See also FileDialog.filters.

PropertyHint **PROPERTY_HINT_INT_IS_OBJECTID** = `29`

Deprecated: This hint is not used by the engine.

PropertyHint **PROPERTY_HINT_INT_IS_POINTER** = `30`

Hints that an int property is a pointer. Used by GDExtension.

PropertyHint **PROPERTY_HINT_ARRAY_TYPE** = `31`

Hints that a property is an Array with the stored type specified in the hint string.

PropertyHint **PROPERTY_HINT_LOCALE_ID** = `32`

Hints that a string property is a locale code. Editing it will show a locale dialog for picking language and country.

PropertyHint **PROPERTY_HINT_LOCALIZABLE_STRING** = `33`

Hints that a dictionary property is string translation map. Dictionary keys are locale codes and, values are translated strings.

PropertyHint **PROPERTY_HINT_NODE_TYPE** = `34`

Hints that a property is an instance of a Node-derived type, optionally specified via the hint string (e.g. `"Node2D"`). Editing it will show a dialog for picking a node from the scene.

PropertyHint **PROPERTY_HINT_HIDE_QUATERNION_EDIT** = `35`

Hints that a quaternion property should disable the temporary euler editor.

PropertyHint **PROPERTY_HINT_PASSWORD** = `36`

Hints that a string property is a password, and every character is replaced with the secret character.

PropertyHint **PROPERTY_HINT_MAX** = `38`

Represents the size of the PropertyHint enum.

---

flags **PropertyUsageFlags**: 🔗

PropertyUsageFlags **PROPERTY_USAGE_NONE** = `0`

The property is not stored, and does not display in the editor. This is the default for non-exported properties.

PropertyUsageFlags **PROPERTY_USAGE_STORAGE** = `2`

The property is serialized and saved in the scene file (default for exported properties).

PropertyUsageFlags **PROPERTY_USAGE_EDITOR** = `4`

The property is shown in the EditorInspector (default for exported properties).

PropertyUsageFlags **PROPERTY_USAGE_INTERNAL** = `8`

The property is excluded from the class reference.

PropertyUsageFlags **PROPERTY_USAGE_CHECKABLE** = `16`

The property can be checked in the EditorInspector.

PropertyUsageFlags **PROPERTY_USAGE_CHECKED** = `32`

The property is checked in the EditorInspector.

PropertyUsageFlags **PROPERTY_USAGE_GROUP** = `64`

Used to group properties together in the editor. See EditorInspector.

PropertyUsageFlags **PROPERTY_USAGE_CATEGORY** = `128`

Used to categorize properties together in the editor.

PropertyUsageFlags **PROPERTY_USAGE_SUBGROUP** = `256`

Used to group properties together in the editor in a subgroup (under a group). See EditorInspector.

PropertyUsageFlags **PROPERTY_USAGE_CLASS_IS_BITFIELD** = `512`

The property is a bitfield, i.e. it contains multiple flags represented as bits.

PropertyUsageFlags **PROPERTY_USAGE_NO_INSTANCE_STATE** = `1024`

The property does not save its state in PackedScene.

PropertyUsageFlags **PROPERTY_USAGE_RESTART_IF_CHANGED** = `2048`

Editing the property prompts the user for restarting the editor.

PropertyUsageFlags **PROPERTY_USAGE_SCRIPT_VARIABLE** = `4096`

The property is a script variable which should be serialized and saved in the scene file.

PropertyUsageFlags **PROPERTY_USAGE_STORE_IF_NULL** = `8192`

The property value of type Object will be stored even if its value is `null`.

PropertyUsageFlags **PROPERTY_USAGE_UPDATE_ALL_IF_MODIFIED** = `16384`

If this property is modified, all inspector fields will be refreshed.

PropertyUsageFlags **PROPERTY_USAGE_SCRIPT_DEFAULT_VALUE** = `32768`

Deprecated: This flag is not used by the engine.

PropertyUsageFlags **PROPERTY_USAGE_CLASS_IS_ENUM** = `65536`

The property is an enum, i.e. it only takes named integer constants from its associated enumeration.

PropertyUsageFlags **PROPERTY_USAGE_NIL_IS_VARIANT** = `131072`

If property has `nil` as default value, its type will be Variant.

PropertyUsageFlags **PROPERTY_USAGE_ARRAY** = `262144`

The property is an array.

PropertyUsageFlags **PROPERTY_USAGE_ALWAYS_DUPLICATE** = `524288`

When duplicating a resource with Resource.duplicate, and this flag is set on a property of that resource, the property should always be duplicated, regardless of the `subresources` bool parameter.

PropertyUsageFlags **PROPERTY_USAGE_NEVER_DUPLICATE** = `1048576`

When duplicating a resource with Resource.duplicate, and this flag is set on a property of that resource, the property should never be duplicated, regardless of the `subresources` bool parameter.

PropertyUsageFlags **PROPERTY_USAGE_HIGH_END_GFX** = `2097152`

The property is only shown in the editor if modern renderers are supported (the Compatibility rendering method is excluded).

PropertyUsageFlags **PROPERTY_USAGE_NODE_PATH_FROM_SCENE_ROOT** = `4194304`

The NodePath property will always be relative to the scene's root. Mostly useful for local resources.

PropertyUsageFlags **PROPERTY_USAGE_RESOURCE_NOT_PERSISTENT** = `8388608`

Use when a resource is created on the fly, i.e. the getter will always return a different instance. ResourceSaver needs this information to properly save such resources.

PropertyUsageFlags **PROPERTY_USAGE_KEYING_INCREMENTS** = `16777216`

Inserting an animation key frame of this property will automatically increment the value, allowing to easily keyframe multiple values in a row.

PropertyUsageFlags **PROPERTY_USAGE_DEFERRED_SET_RESOURCE** = `33554432`

Deprecated: This flag is not used by the engine.

PropertyUsageFlags **PROPERTY_USAGE_EDITOR_INSTANTIATE_OBJECT** = `67108864`

When this property is a Resource and base object is a Node, a resource instance will be automatically created whenever the node is created in the editor.

PropertyUsageFlags **PROPERTY_USAGE_EDITOR_BASIC_SETTING** = `134217728`

The property is considered a basic setting and will appear even when advanced mode is disabled. Used for project settings.

PropertyUsageFlags **PROPERTY_USAGE_READ_ONLY** = `268435456`

The property is read-only in the EditorInspector.

PropertyUsageFlags **PROPERTY_USAGE_SECRET** = `536870912`

An export preset property with this flag contains confidential information and is stored separately from the rest of the export preset configuration.

PropertyUsageFlags **PROPERTY_USAGE_DEFAULT** = `6`

Default usage (storage and editor).

PropertyUsageFlags **PROPERTY_USAGE_NO_EDITOR** = `2`

Default usage but without showing the property in the editor (storage).

flags **MethodFlags**: 🔗

MethodFlags **METHOD_FLAG_NORMAL** = `1`

Flag for a normal method.

MethodFlags **METHOD_FLAG_EDITOR** = `2`

Flag for an editor method.

MethodFlags **METHOD_FLAG_CONST** = `4`

Flag for a constant method.

MethodFlags **METHOD_FLAG_VIRTUAL** = `8`

Flag for a virtual method.

MethodFlags **METHOD_FLAG_VARARG** = `16`

Flag for a method with a variable number of arguments.

MethodFlags **METHOD_FLAG_STATIC** = `32`

Flag for a static method.

MethodFlags **METHOD_FLAG_OBJECT_CORE** = `64`

Used internally. Allows to not dump core virtual methods (such as Object._notification) to the JSON API.

MethodFlags **METHOD_FLAGS_DEFAULT** = `1`

Default method flags (normal).

---

enum **Variant.Type**: 🔗

Variant.Type **TYPE_NIL** = `0`

Variable is `null`.

Variant.Type **TYPE_BOOL** = `1`

Variable is of type bool.

Variant.Type **TYPE_INT** = `2`

Variable is of type int.

Variant.Type **TYPE_FLOAT** = `3`

Variable is of type float.

Variant.Type **TYPE_STRING** = `4`

Variable is of type String.

Variant.Type **TYPE_VECTOR2** = `5`

Variable is of type Vector2.

Variant.Type **TYPE_VECTOR2I** = `6`

Variable is of type Vector2i.

Variant.Type **TYPE_RECT2** = `7`

Variable is of type Rect2.

Variant.Type **TYPE_RECT2I** = `8`

Variable is of type Rect2i.

Variant.Type **TYPE_VECTOR3** = `9`

Variable is of type Vector3.

Variant.Type **TYPE_VECTOR3I** = `10`

Variable is of type Vector3i.

Variant.Type **TYPE_TRANSFORM2D** = `11`

Variable is of type Transform2D.

Variant.Type **TYPE_VECTOR4** = `12`

Variable is of type Vector4.

Variant.Type **TYPE_VECTOR4I** = `13`

Variable is of type Vector4i.

Variant.Type **TYPE_PLANE** = `14`

Variable is of type Plane.

Variant.Type **TYPE_QUATERNION** = `15`

Variable is of type Quaternion.

Variant.Type **TYPE_AABB** = `16`

Variable is of type AABB.

Variant.Type **TYPE_BASIS** = `17`

Variable is of type Basis.

Variant.Type **TYPE_TRANSFORM3D** = `18`

Variable is of type Transform3D.

Variant.Type **TYPE_PROJECTION** = `19`

Variable is of type Projection.

Variant.Type **TYPE_COLOR** = `20`

Variable is of type Color.

Variant.Type **TYPE_STRING_NAME** = `21`

Variable is of type StringName.

Variant.Type **TYPE_NODE_PATH** = `22`

Variable is of type NodePath.

Variant.Type **TYPE_RID** = `23`

Variable is of type RID.

Variant.Type **TYPE_OBJECT** = `24`

Variable is of type Object.

Variant.Type **TYPE_CALLABLE** = `25`

Variable is of type Callable.

Variant.Type **TYPE_SIGNAL** = `26`

Variable is of type Signal.

Variant.Type **TYPE_DICTIONARY** = `27`

Variable is of type Dictionary.

Variant.Type **TYPE_ARRAY** = `28`

Variable is of type Array.

Variant.Type **TYPE_PACKED_BYTE_ARRAY** = `29`

Variable is of type PackedByteArray.

Variant.Type **TYPE_PACKED_INT32_ARRAY** = `30`

Variable is of type PackedInt32Array.

Variant.Type **TYPE_PACKED_INT64_ARRAY** = `31`

Variable is of type PackedInt64Array.

Variant.Type **TYPE_PACKED_FLOAT32_ARRAY** = `32`

Variable is of type PackedFloat32Array.

Variant.Type **TYPE_PACKED_FLOAT64_ARRAY** = `33`

Variable is of type PackedFloat64Array.

Variant.Type **TYPE_PACKED_STRING_ARRAY** = `34`

Variable is of type PackedStringArray.

Variant.Type **TYPE_PACKED_VECTOR2_ARRAY** = `35`

Variable is of type PackedVector2Array.

Variant.Type **TYPE_PACKED_VECTOR3_ARRAY** = `36`

Variable is of type PackedVector3Array.

Variant.Type **TYPE_PACKED_COLOR_ARRAY** = `37`

Variable is of type PackedColorArray.

Variant.Type **TYPE_PACKED_VECTOR4_ARRAY** = `38`

Variable is of type PackedVector4Array.

Variant.Type **TYPE_MAX** = `39`

Represents the size of the Variant.Type enum.

---

enum **Variant.Operator**: 🔗

Variant.Operator **OP_EQUAL** = `0`

Equality operator (`==`).

Variant.Operator **OP_NOT_EQUAL** = `1`

Inequality operator (`!=`).

Variant.Operator **OP_LESS** = `2`

Less than operator (`<`).

Variant.Operator **OP_LESS_EQUAL** = `3`

Less than or equal operator (`<=`).

Variant.Operator **OP_GREATER** = `4`

Greater than operator (`>`).

Variant.Operator **OP_GREATER_EQUAL** = `5`

Greater than or equal operator (`>=`).

Variant.Operator **OP_ADD** = `6`

Addition operator (`+`).

Variant.Operator **OP_SUBTRACT** = `7`

Subtraction operator (`-`).

Variant.Operator **OP_MULTIPLY** = `8`

Multiplication operator (`*`).

Variant.Operator **OP_DIVIDE** = `9`

Division operator (`/`).

Variant.Operator **OP_NEGATE** = `10`

Unary negation operator (`-`).

Variant.Operator **OP_POSITIVE** = `11`

Unary plus operator (`+`).

Variant.Operator **OP_MODULE** = `12`

Remainder/modulo operator (`%`).

Variant.Operator **OP_POWER** = `13`

Power operator (`**`).

Variant.Operator **OP_SHIFT_LEFT** = `14`

Left shift operator (`<<`).

Variant.Operator **OP_SHIFT_RIGHT** = `15`

Right shift operator (`>>`).

Variant.Operator **OP_BIT_AND** = `16`

Bitwise AND operator (`&`).

Variant.Operator **OP_BIT_OR** = `17`

Bitwise OR operator (`|`).

Variant.Operator **OP_BIT_XOR** = `18`

Bitwise XOR operator (`^`).

Variant.Operator **OP_BIT_NEGATE** = `19`

Bitwise NOT operator (`~`).

Variant.Operator **OP_AND** = `20`

Logical AND operator (`and` or `&&`).

Variant.Operator **OP_OR** = `21`

Logical OR operator (`or` or `||`).

Variant.Operator **OP_XOR** = `22`

Logical XOR operator (not implemented in GDScript).

Variant.Operator **OP_NOT** = `23`

Logical NOT operator (`not` or `!`).

Variant.Operator **OP_IN** = `24`

Logical IN operator (`in`).

Variant.Operator **OP_MAX** = `25`

Represents the size of the Variant.Operator enum.

Property Descriptions

AudioServer **AudioServer**

The AudioServer singleton.

CameraServer **CameraServer**

The CameraServer singleton.

ClassDB **ClassDB**

The ClassDB singleton.

DisplayServer **DisplayServer**

The DisplayServer singleton.

EditorInterface **EditorInterface** 🔗

The EditorInterface singleton.

Note: Only available in editor builds.

Engine **Engine** 🔗

The Engine singleton.

EngineDebugger **EngineDebugger** 🔗

The EngineDebugger singleton.

GDExtensionManager **GDExtensionManager** 🔗

The GDExtensionManager singleton.

Geometry2D **Geometry2D** 🔗

The Geometry2D singleton.

Geometry3D **Geometry3D** 🔗

The Geometry3D singleton.

IP **IP** 🔗

The IP singleton.

Input **Input** 🔗

The Input singleton.

InputMap **InputMap** 🔗

The InputMap singleton.

JavaClassWrapper **JavaClassWrapper** 🔗

The JavaClassWrapper singleton.

Note: Only implemented on Android.

JavaScriptBridge **JavaScriptBridge** 🔗

The JavaScriptBridge singleton.

Note: Only implemented on the Web platform.

Marshalls **Marshalls** 🔗

The Marshalls singleton.

NativeMenu **NativeMenu** 🔗

The NativeMenu singleton.

Note: Only implemented on macOS.

NavigationMeshGenerator **NavigationMeshGenerator** 🔗

The NavigationMeshGenerator singleton.

NavigationServer2D **NavigationServer2D**

The NavigationServer2D singleton.

NavigationServer3D **NavigationServer3D**

The NavigationServer3D singleton.

OS **OS**

The OS singleton.

Performance **Performance**

The Performance singleton.

PhysicsServer2D **PhysicsServer2D**

The PhysicsServer2D singleton.

PhysicsServer2DManager **PhysicsServer2DManager**

The PhysicsServer2DManager singleton.

PhysicsServer3D **PhysicsServer3D**

The PhysicsServer3D singleton.

PhysicsServer3DManager **PhysicsServer3DManager** 🔗

The PhysicsServer3DManager singleton.

ProjectSettings **ProjectSettings** 🔗

The ProjectSettings singleton.

RenderingServer **RenderingServer** 🔗

The RenderingServer singleton.

ResourceLoader **ResourceLoader** 🔗

The ResourceLoader singleton.

ResourceSaver **ResourceSaver** 🔗

The ResourceSaver singleton.

ResourceUID **ResourceUID** 🔗

The ResourceUID singleton.

TextServerManager **TextServerManager** 🔗

The TextServerManager singleton.

ThemeDB **ThemeDB** 🔗

The ThemeDB singleton.

Time **Time**

The Time singleton.

TranslationServer **TranslationServer**

The TranslationServer singleton.

WorkerThreadPool **WorkerThreadPool**

The WorkerThreadPool singleton.

XRServer **XRServer**

The XRServer singleton.

Method Descriptions

Variant **abs**(x: Variant)

Returns the absolute value of a Variant parameter `x` (i.e. non-negative value). Supported types: int, float, Vector2, Vector2i, Vector3, Vector3i, Vector4, Vector4i.

```
var a = abs(-1)
# a is 1

var b = abs(-1.2)
# b is 1.2
```

```
var c = abs(Vector2(-3.5, -4))
# c is (3.5, 4)

var d = abs(Vector2i(-5, -6))
# d is (5, 6)

var e = abs(Vector3(-7, 8.5, -3.8))
# e is (7, 8.5, 3.8)

var f = abs(Vector3i(-7, -8, -9))
# f is (7, 8, 9)
```

Note: For better type safety, use absf, absi, Vector2.abs, Vector2i.abs, Vector3.abs, Vector3i.abs, Vector4.abs, or Vector4i.abs.

---

float **absf**(x: float)

Returns the absolute value of float parameter `x` (i.e. positive value).

```
# a is 1.2
var a = absf(-1.2)
```

---

int **absi**(x: int)

Returns the absolute value of int parameter `x` (i.e. positive value).

```
# a is 1
var a = absi(-1)
```

---

float **acos**(x: float)

Returns the arc cosine of `x` in radians. Use to get the angle of cosine `x`. `x` will be clamped between `-1.0` and `1.0` (inclusive), in order to prevent acos from returning @GDScript.NAN.

```
# c is 0.523599 or 30 degrees if converted with rad_to_deg(c)
var c = acos(0.866025)
```

---

float **acosh**(x: float) 🔗

Returns the hyperbolic arc (also called inverse) cosine of `x`, returning a value in radians. Use it to get the angle from an angle's cosine in hyperbolic space if `x` is larger or equal to 1. For values of `x` lower than 1, it will return 0, in order to prevent acosh from returning @GDScript.NAN.

```
var a = acosh(2) # Returns 1.3169578969248
cosh(a) # Returns 2

var b = acosh(-1) # Returns 0
```

---

float **angle_difference**(from: float, to: float) 🔗

Returns the difference between the two angles, in the range of `[-PI, +PI]`. When `from` and `to` are opposite, returns `-PI` if `from` is smaller than `to`, or `PI` otherwise.

---

float **asin**(x: float) 🔗

Returns the arc sine of `x` in radians. Use to get the angle of sine `x`. `x` will be clamped between `-1.0` and `1.0` (inclusive), in order to prevent asin from returning @GDScript.NAN.

```
# s is 0.523599 or 30 degrees if converted with rad_to_deg(s)
var s = asin(0.5)
```

---

float **asinh**(x: float) 🔗

Returns the hyperbolic arc (also called inverse) sine of `x`, returning a value in radians. Use it to get the angle from an angle's sine in hyperbolic space.

```
var a = asinh(0.9) # Returns 0.8088669356527824
sinh(a) # Returns 0.9
```

float **atan**(x: float)

Returns the arc tangent of `x` in radians. Use it to get the angle from an angle's tangent in trigonometry.

The method cannot know in which quadrant the angle should fall. See atan2 if you have both `y` and `x`.

```
var a = atan(0.5) # a is 0.463648
```

If `x` is between `-PI / 2` and `PI / 2` (inclusive), `atan(tan(x))` is equal to `x`.

float **atan2**(y: float, x: float)

Returns the arc tangent of `y/x` in radians. Use to get the angle of tangent `y/x`. To compute the value, the method takes into account the sign of both arguments in order to determine the quadrant.

Important note: The Y coordinate comes first, by convention.

```
var a = atan2(0, -1) # a is 3.141593
```

float **atanh**(x: float)

Returns the hyperbolic arc (also called inverse) tangent of `x`, returning a value in radians. Use it to get the angle from an angle's tangent in hyperbolic space if `x` is between -1 and 1 (non-inclusive).

In mathematics, the inverse hyperbolic tangent is only defined for -1 < `x` < 1 in the real set, so values equal or lower to -1 for `x` return negative @GDScript.INF and values equal or higher than 1 return positive @GDScript.INF in order to prevent atanh from returning @GDScript.NAN.

```
var a = atanh(0.9) # Returns 1.47221948958322
tanh(a) # Returns 0.9

var b = atanh(-2) # Returns -inf
tanh(b) # Returns -1
```

---

float **bezier_derivative**(start: float, control_1: float, control_2: float, end: float, t: float) 🔗

Returns the derivative at the given `t` on a one-dimensional Bézier curve [https://en.wikipedia.org/wiki/B%C3%A9zier_curve] defined by the given `control_1`, `control_2`, and `end` points.

---

float **bezier_interpolate**(start: float, control_1: float, control_2: float, end: float, t: float) 🔗

Returns the point at the given `t` on a one-dimensional Bézier curve [https://en.wikipedia.org/wiki/B%C3%A9zier_curve] defined by the given `control_1`, `control_2`, and `end` points.

Variant **bytes_to_var**(bytes: PackedByteArray) 🔗

Decodes a byte array back to a Variant value, without decoding objects.

Note: If you need object deserialization, see bytes_to_var_with_objects.

Variant **bytes_to_var_with_objects**(bytes: PackedByteArray)

Decodes a byte array back to a Variant value. Decoding objects is allowed.

Warning: Deserialized object can contain code which gets executed. Do not use this option if the serialized object comes from untrusted sources to avoid potential security threats (remote code execution).

Variant **ceil**(x: Variant)

Rounds `x` upward (towards positive infinity), returning the smallest whole number that is not less than `x`. Supported types: int, float, Vector2, Vector2i, Vector3, Vector3i, Vector4, Vector4i.

```
var i = ceil(1.45) # i is 2.0
i = ceil(1.001)    # i is 2.0
```

See also floor, round, and snapped.

Note: For better type safety, use ceilf, ceili, Vector2.ceil, Vector3.ceil, or Vector4.ceil.

---

float **ceilf**(x: float)

Rounds `x` upward (towards positive infinity), returning the smallest whole number that is not less than `x`.

A type-safe version of ceil, returning a float.

---

int **ceili**(x: float)

Rounds `x` upward (towards positive infinity), returning the smallest whole number that is not less than `x`.

A type-safe version of ceil, returning an int.

Variant **clamp**(value: Variant, min: Variant, max: Variant)

Clamps the `value`, returning a Variant not less than `min` and not more than `max`. Any values that can be compared with the less than and greater than operators will work.

```
var a = clamp(-10, -1, 5)
# a is -1

var b = clamp(8.1, 0.9, 5.5)
# b is 5.5
```

Note: For better type safety, use clampf, clampi, Vector2.clamp, Vector2i.clamp, Vector3.clamp, Vector3i.clamp, Vector4.clamp, Vector4i.clamp, or Color.clamp (not currently supported by this method).

Note: When using this on vectors it will *not* perform component-wise clamping, and will pick `min` if `value < min` or `max` if `value > max`. To perform component-wise clamping use the methods listed above.

---

float **clampf**(value: float, min: float, max: float)

Clamps the `value`, returning a float not less than `min` and not more than `max`.

```
var speed = 42.1
var a = clampf(speed, 1.0, 20.5) # a is 20.5
```

```
speed = -10.0
var b = clampf(speed, -1.0, 1.0) # b is -1.0
```

---

int **clampi**(value: int, min: int, max: int) 🔗

Clamps the `value`, returning an int not less than `min` and not more than `max`.

```
var speed = 42
var a = clampi(speed, 1, 20) # a is 20

speed = -10
var b = clampi(speed, -1, 1) # b is -1
```

---

float **cos**(angle_rad: float) 🔗

Returns the cosine of angle `angle_rad` in radians.

```
cos(PI * 2)         # Returns 1.0
cos(PI)             # Returns -1.0
cos(deg_to_rad(90)) # Returns 0.0
```

---

float **cosh**(x: float) 🔗

Returns the hyperbolic cosine of `x` in radians.

```
print(cosh(1)) # Prints 1.543081
```

---

float **cubic_interpolate**(from: float, to: float, pre: float, post: float, weight: float) 🔗

Cubic interpolates between two values by the factor defined in `weight` with `pre` and `post` values.

---

float **cubic_interpolate_angle**(from: float, to: float, pre: float, post: float, weight: float) 🔗

Cubic interpolates between two rotation values with shortest path by the factor defined in `weight` with `pre` and `post` values. See also lerp_angle.

---

float **cubic_interpolate_angle_in_time**(from: float, to: float, pre: float, post: float, weight: float, to_t: float, pre_t: float, post_t: float) 🔗

Cubic interpolates between two rotation values with shortest path by the factor defined in `weight` with `pre` and `post` values. See also lerp_angle.

It can perform smoother interpolation than cubic_interpolate by the time values.

---

float **cubic_interpolate_in_time**(from: float, to: float, pre: float, post: float, weight: float, to_t: float, pre_t: float, post_t: float) 🔗

Cubic interpolates between two values by the factor defined in `weight` with `pre` and `post` values.

It can perform smoother interpolation than cubic_interpolate by the time values.

---

float **db_to_linear**(db: float) 🔗

Converts from decibels to linear energy (audio).

---

float **deg_to_rad**(deg: float) 🔗

Converts an angle expressed in degrees to radians.

```
var r = deg_to_rad(180) # r is 3.141593
```

---

float **ease**(x: float, curve: float)

Returns an "eased" value of `x` based on an easing function defined with `curve`. This easing function is based on an exponent. The `curve` can be any floating-point number, with specific values leading to the following behaviors:

```
- Lower than -1.0 (exclusive): Ease in-out
- 1.0: Linear
- Between -1.0 and 0.0 (exclusive): Ease out-in
- 0.0: Constant
- Between 0.0 to 1.0 (exclusive): Ease out
- 1.0: Linear
- Greater than 1.0 (exclusive): Ease in
```

ease() curve values cheatsheet
[https://raw.githubusercontent.com/godotengine/godot-docs/master/img/ease_cheatsheet.png]

See also smoothstep. If you need to perform more advanced transitions, use Tween.interpolate_value.

String **error_string**(error: int)

Returns a human-readable name for the given Error code.

```
print(OK)                               # Prints 0
print(error_string(OK))                 # Prints OK
print(error_string(ERR_BUSY))           # Prints Busy
print(error_string(ERR_OUT_OF_MEMORY)) # Prints Out of memory
```

---

float **exp**(x: float)

The natural exponential function. It raises the mathematical constant *e* to the power of `x` and returns it.

e has an approximate value of 2.71828, and can be obtained with `exp(1)`.

For exponents to other bases use the method pow.

```
var a = exp(2) # Approximately 7.39
```

Variant **floor**(x: Variant)

Rounds `x` downward (towards negative infinity), returning the largest whole number that is not more than `x`. Supported types: int, float, Vector2, Vector2i, Vector3, Vector3i, Vector4, Vector4i.

```
var a = floor(2.99) # a is 2.0
a = floor(-2.99)    # a is -3.0
```

See also ceil, round, and snapped.

Note: For better type safety, use floorf, floori, Vector2.floor, Vector3.floor, or Vector4.floor.

---

float **floorf**(x: float)

Rounds `x` downward (towards negative infinity), returning the largest whole number that is not more than `x`.

A type-safe version of floor, returning a float.

---

int **floori**(x: float)

Rounds `x` downward (towards negative infinity), returning the largest whole number that is not more than `x`.

A type-safe version of floor, returning an int.

Note: This function is *not* the same as `int(x)`, which rounds towards 0.

---

float **fmod**(x: float, y: float)

Returns the floating-point remainder of `x` divided by `y`, keeping the sign of `x`.

```
var remainder = fmod(7, 5.5) # remainder is 1.5
```

For the integer remainder operation, use the `%` operator.

---

float **fposmod**(x: float, y: float)

Returns the floating-point modulus of `x` divided by `y`, wrapping equally in positive and negative.

```
print(" (x)  (fmod(x, 1.5))   (fposmod(x, 1.5))")
for i in 7:
    var x = i * 0.5 - 1.5
    print("%4.1f           %4.1f  | %4.1f" % [x, fmod(x, 1.5),
fposmod(x, 1.5)])
```

Prints:

```
 (x)  (fmod(x, 1.5))   (fposmod(x, 1.5))
-1.5           -0.0  |  0.0
-1.0           -1.0  |  0.5
-0.5           -0.5  |  1.0
 0.0            0.0  |  0.0
 0.5            0.5  |  0.5
 1.0            1.0  |  1.0
 1.5            0.0  |  0.0
```

---

int **hash**(variable: Variant) 🔗

Returns the integer hash of the passed `variable`.

GDScript

```
print(hash("a")) # Prints 177670
```

C#

```
GD.Print(GD.Hash("a")); // Prints 177670
```

Object **instance_from_id**(instance_id: int) 🔗

Returns the Object that corresponds to `instance_id`. All Objects have a unique instance ID. See also Object.get_instance_id.

GDScript

```
var foo = "bar"

func _ready():
    var id = get_instance_id()
    var inst = instance_from_id(id)
    print(inst.foo) # Prints bar
```

C#

```
public partial class MyNode : Node
{
    public string Foo { get; set; } = "bar";

    public override void _Ready()
    {
        ulong id = GetInstanceId();
        var inst = (MyNode)InstanceFromId(Id);
        GD.Print(inst.Foo); // Prints bar
    }
}
```

---

float **inverse_lerp**(from: float, to: float, weight: float)

Returns an interpolation or extrapolation factor considering the range specified in `from` and `to`, and the interpolated value specified in `weight`. The returned value will be between `0.0` and `1.0` if `weight` is between `from` and `to` (inclusive). If `weight` is located outside this range, then an extrapolation factor will be returned (return value lower than `0.0` or greater than `1.0`). Use clamp on the result of inverse_lerp if this is not desired.

```
# The interpolation ratio in the `lerp()` call below is 0.75.
var middle = lerp(20, 30, 0.75)
# middle is now 27.5.

# Now, we pretend to have forgotten the original ratio and want to get it back.
var ratio = inverse_lerp(20, 30, 27.5)
# ratio is now 0.75.
```

See also lerp, which performs the reverse of this operation, and remap to map a continuous series of values to another.

---

bool **is_equal_approx**(a: float, b: float)

Returns `true` if `a` and `b` are approximately equal to each other.

Here, "approximately equal" means that `a` and `b` are within a small internal epsilon of each other, which scales with the magnitude of the numbers.

Infinity values of the same sign are considered equal.

---

bool **is_finite**(x: float)

Returns whether `x` is a finite value, i.e. it is not @GDScript.NAN, positive infinity, or negative infinity.

bool **is_inf**(x: float)

Returns `true` if `x` is either positive infinity or negative infinity.

bool **is_instance_id_valid**(id: int)

Returns `true` if the Object that corresponds to `id` is a valid object (e.g. has not been deleted from memory). All Objects have a unique instance ID.

bool **is_instance_valid**(instance: Variant)

Returns `true` if `instance` is a valid Object (e.g. has not been deleted from memory).

bool **is_nan**(x: float)

Returns `true` if `x` is a NaN ("Not a Number" or invalid) value.

bool **is_same**(a: Variant, b: Variant)

Returns `true`, for value types, if `a` and `b` share the same value. Returns `true`, for reference types, if the references of `a` and `b` are the same.

```
# Vector2 is a value type
var vec2_a = Vector2(0, 0)
var vec2_b = Vector2(0, 0)
var vec2_c = Vector2(1, 1)
is_same(vec2_a, vec2_a)  # true
is_same(vec2_a, vec2_b)  # true
is_same(vec2_a, vec2_c)  # false

# Array is a reference type
var arr_a = []
var arr_b = []
is_same(arr_a, arr_a)  # true
is_same(arr_a, arr_b)  # false
```

These are Variant value types: `null`, bool, int, float, String, StringName, Vector2, Vector2i, Vector3, Vector3i, Vector4, Vector4i, Rect2, Rect2i, Transform2D, Transform3D, Plane, Quaternion, AABB, Basis, Projection, Color, NodePath, RID, Callable and Signal.

These are Variant reference types: Object, Dictionary, Array, PackedByteArray, PackedInt32Array, PackedInt64Array, PackedFloat32Array, PackedFloat64Array, PackedStringArray, PackedVector2Array, PackedVector3Array, PackedVector4Array, and PackedColorArray.

bool **is_zero_approx**(x: float) 🔗

Returns `true` if `x` is zero or almost zero. The comparison is done using a tolerance calculation with a small internal epsilon.

This function is faster than using is_equal_approx with one value as zero.

Variant **lerp**(from: Variant, to: Variant, weight: Variant) 🔗

Linearly interpolates between two values by the factor defined in `weight`. To perform interpolation, `weight` should be between `0.0` and

`1.0` (inclusive). However, values outside this range are allowed and can be used to perform *extrapolation*. If this is not desired, use clamp on the result of this function.

Both `from` and `to` must be the same type. Supported types: int, float, Vector2, Vector3, Vector4, Color, Quaternion, Basis.

```
lerp(0, 4, 0.75) # Returns 3.0
```

See also inverse_lerp which performs the reverse of this operation. To perform eased interpolation with lerp, combine it with ease or smoothstep. See also remap to map a continuous series of values to another.

Note: For better type safety, use lerpf, Vector2.lerp, Vector3.lerp, Vector4.lerp, Color.lerp, Quaternion.slerp or Basis.slerp.

---

float **lerp_angle**(from: float, to: float, weight: float) 🔗

Linearly interpolates between two angles (in radians) by a `weight` value between 0.0 and 1.0.

Similar to lerp, but interpolates correctly when the angles wrap around @GDScript.TAU. To perform eased interpolation with lerp_angle, combine it with ease or smoothstep.

```
extends Sprite
var elapsed = 0.0
func _process(delta):
    var min_angle = deg_to_rad(0.0)
    var max_angle = deg_to_rad(90.0)
    rotation = lerp_angle(min_angle, max_angle, elapsed)
    elapsed += delta
```

Note: This function lerps through the shortest path between `from` and `to`. However, when these two angles are approximately `PI + k * TAU` apart for any integer `k`, it's not obvious which way they lerp due to floating-point precision errors. For example, `lerp_angle(0, PI, weight)` lerps counter-clockwise, while `lerp_angle(0, PI + 5 * TAU, weight)` lerps clockwise.

---

float **lerpf**(from: float, to: float, weight: float)

Linearly interpolates between two values by the factor defined in `weight`. To perform interpolation, `weight` should be between `0.0` and `1.0` (inclusive). However, values outside this range are allowed and can be used to perform *extrapolation*. If this is not desired, use clampf on the result of this function.

```
lerpf(0, 4, 0.75) # Returns 3.0
```

See also inverse_lerp which performs the reverse of this operation. To perform eased interpolation with lerp, combine it with ease or smoothstep.

---

float **linear_to_db**(lin: float)

Converts from linear energy to decibels (audio). This can be used to implement volume sliders that behave as expected (since volume isn't linear).

Example:

```
# "Slider" refers to a node that inherits Range such as HSlider
or VSlider.
# Its range must be configured to go from 0 to 1.
# Change the bus name if you'd like to change the volume of a
specific bus only.
AudioServer.set_bus_volume_db(AudioServer.get_bus_index("Master
"), linear_to_db($Slider.value))
```

---

float **log**(x: float)

Returns the natural logarithm [https://en.wikipedia.org/wiki/Natural_logarithm] of `x` (base [i]e[/i] [https://en.wikipedia.org/wiki/E_(mathematical_constant)], with *e* being approximately 2.71828). This is the amount of time needed to reach a certain level of continuous growth.

Note: This is not the same as the "log" function on most calculators, which uses a base 10 logarithm. To use base 10 logarithm, use `log(x) / log(10)`.

```
log(10) # Returns 2.302585
```

Note: The logarithm of `0` returns `-inf`, while negative values return `-nan`.

Variant **max**(...) vararg 🔗

Returns the maximum of the given numeric values. This function can take any number of arguments.

```
max(1, 7, 3, -6, 5) # Returns 7
```

Note: When using this on vectors it will *not* perform component-wise maximum, and will pick the largest value when compared using `x < y`. To perform component-wise maximum, use Vector2.max, Vector2i.max, Vector3.max, Vector3i.max, Vector4.max, and Vector4i.max.

---

float **maxf**(a: float, b: float) 🔗

Returns the maximum of two float values.

```
maxf(3.6, 24)    # Returns 24.0
maxf(-3.99, -4) # Returns -3.99
```

---

int **maxi**(a: int, b: int)

Returns the maximum of two int values.

```
maxi(1, 2)    # Returns 2
maxi(-3, -4) # Returns -3
```

Variant **min**(...) vararg

Returns the minimum of the given numeric values. This function can take any number of arguments.

```
min(1, 7, 3, -6, 5) # Returns -6
```

Note: When using this on vectors it will *not* perform component-wise minimum, and will pick the smallest value when compared using `x < y`. To perform component-wise minimum, use Vector2.min, Vector2i.min, Vector3.min, Vector3i.min, Vector4.min, and Vector4i.min.

float **minf**(a: float, b: float)

Returns the minimum of two float values.

```
minf(3.6, 24)    # Returns 3.6
minf(-3.99, -4) # Returns -4.0
```

int **mini**(a: int, b: int)

Returns the minimum of two int values.

```
mini(1, 2)    # Returns 1
mini(-3, -4) # Returns -4
```

float **move_toward**(from: float, to: float, delta: float)

Moves `from` toward `to` by the `delta` amount. Will not go past `to`.

Use a negative `delta` value to move away.

```
move_toward(5, 10, 4)     # Returns 9
move_toward(10, 5, 4)     # Returns 6
move_toward(5, 10, 9)     # Returns 10
move_toward(10, 5, -1.5) # Returns 11.5
```

int **nearest_po2**(value: int)

Returns the smallest integer power of 2 that is greater than or equal to `value`.

```
nearest_po2(3) # Returns 4
nearest_po2(4) # Returns 4
nearest_po2(5) # Returns 8

nearest_po2(0)  # Returns 0 (this may not be expected)
nearest_po2(-1) # Returns 0 (this may not be expected)
```

Warning: Due to its implementation, this method returns `0` rather than `1` for values less than or equal to `0`, with an exception for `value` being the smallest negative 64-bit integer (`-9223372036854775808`) in which case the `value` is returned unchanged.

float **pingpong**(value: float, length: float)

Wraps `value` between `0` and the `length`. If the limit is reached, the next value the function returns is decreased to the `0` side or increased to the `length` side (like a triangle wave). If `length` is less than zero, it becomes positive.

```
pingpong(-3.0, 3.0) # Returns 3.0
pingpong(-2.0, 3.0) # Returns 2.0
pingpong(-1.0, 3.0) # Returns 1.0
pingpong(0.0, 3.0)  # Returns 0.0
pingpong(1.0, 3.0)  # Returns 1.0
pingpong(2.0, 3.0)  # Returns 2.0
pingpong(3.0, 3.0)  # Returns 3.0
pingpong(4.0, 3.0)  # Returns 2.0
pingpong(5.0, 3.0)  # Returns 1.0
pingpong(6.0, 3.0)  # Returns 0.0
```

---

int **posmod**(x: int, y: int)

Returns the integer modulus of `x` divided by `y` that wraps equally in positive and negative.

```
print("#(i)  (i % 3)   (posmod(i, 3))")
for i in range(-3, 4):
    print("%2d       %2d  | %2d" % [i, i % 3, posmod(i, 3)])
```

Prints:

```
(i)  (i % 3)   (posmod(i, 3))
-3        0  |  0
-2       -2  |  1
-1       -1  |  2
 0        0  |  0
 1        1  |  1
 2        2  |  2
 3        0  |  0
```

---

float **pow**(base: float, exp: float)

Returns the result of `base` raised to the power of `exp`.

In GDScript, this is the equivalent of the `**` operator.

```
pow(2, 5)   # Returns 32.0
pow(4, 1.5) # Returns 8.0
```

---

void **print**(...) vararg

Converts one or more arguments of any type to string in the best way possible and prints them to the console.

GDScript

```
var a = [1, 2, 3]
print("a", "b", a) # Prints ab[1, 2, 3]
```

C#

```
var a = new Godot.Collections.Array { 1, 2, 3 };
GD.Print("a", "b", a); // Prints ab[1, 2, 3]
```

Note: Consider using push_error and push_warning to print error and warning messages instead of print or print_rich. This distinguishes them from print messages used for debugging purposes, while also displaying a stack trace when an error or warning is printed.

---

void **print_rich**(...) vararg

Converts one or more arguments of any type to string in the best way possible and prints them to the console.

The following BBCode tags are supported: `b`, `i`, `u`, `s`, `indent`, `code`, `url`, `center`, `right`, `color`, `bgcolor`, `fgcolor`.

Color tags only support the following named colors: `black`, `red`, `green`, `yellow`, `blue`, `magenta`, `pink`, `purple`, `cyan`, `white`, `orange`, `gray`. Hexadecimal color codes are not supported.

URL tags only support URLs wrapped by a URL tag, not URLs with a different title.

When printing to standard output, the supported subset of BBCode is converted to ANSI escape codes for the terminal emulator to display. Support for ANSI escape codes varies across terminal emulators, especially for italic and strikethrough. In standard output, `code` is represented with faint text but without any font change. Unsupported tags are left as-is in standard output.

GDScript

```
print_rich("[color=green][b]Hello world![/b][/color]") # Prints
out "Hello world!" in green with a bold font
```

C#

```
GD.PrintRich("[color=green][b]Hello world![/b][/color]"); //
Prints out "Hello world!" in green with a bold font
```

Note: Consider using push_error and push_warning to print error and warning messages instead of print or print_rich. This distinguishes them from print messages used for debugging purposes, while also displaying a stack trace when an error or warning is printed.

Note: On Windows, only Windows 10 and later correctly displays ANSI escape codes in standard output.

Note: Output displayed in the editor supports clickable `[url=address]text[/url]` tags. The `[url]` tag's `address` value is handled by OS.shell_open when clicked.

---

void **print_verbose**(...) vararg 🔗

If verbose mode is enabled (OS.is_stdout_verbose returning `true`), converts one or more arguments of any type to string in the best way possible and prints them to the console.

---

void **printerr**(...) vararg 🔗

Prints one or more arguments to strings in the best way possible to standard error line.

GDScript

```
printerr("prints to stderr")
```

C#

```
GD.PrintErr("prints to stderr");
```

---

void **printraw**(...) vararg 🔗

Prints one or more arguments to strings in the best way possible to the OS terminal. Unlike print, no newline is automatically added at the end.

Note: The OS terminal is *not* the same as the editor's Output dock. The output sent to the OS terminal can be seen when running Godot from a terminal. On Windows, this requires using the `console.exe` executable.

GDScript

```
printraw("A")
printraw("B")
printraw("C")
# Prints ABC to terminal
```

C#

```
GD.PrintRaw("A");
GD.PrintRaw("B");
GD.PrintRaw("C");
// Prints ABC to terminal
```

---

void **prints**(...) vararg 🔗

Prints one or more arguments to the console with a space between each argument.

GDScript

```
prints("A", "B", "C") # Prints A B C
```

C#

```
GD.PrintS("A", "B", "C"); // Prints A B C
```

void **printt**(...) vararg

Prints one or more arguments to the console with a tab between each argument.

GDScript

```
printt("A", "B", "C") # Prints A       B       C
```

C#

```
GD.PrintT("A", "B", "C"); // Prints A       B       C
```

void **push_error**(...) vararg

Pushes an error message to Godot's built-in debugger and to the OS terminal.

GDScript

```
push_error("test error") # Prints "test error" to debugger and
terminal as error call
```

C#

```
GD.PushError("test error"); // Prints "test error" to debugger
and terminal as error call
```

Note: This function does not pause project execution. To print an error message and pause project execution in debug builds, use `assert(false, "test error")` instead.

---

void **push_warning**(...) vararg 🔗

Pushes a warning message to Godot's built-in debugger and to the OS terminal.

GDScript

```
push_warning("test warning") # Prints "test warning" to
debugger and terminal as warning call
```

C#

```
GD.PushWarning("test warning"); // Prints "test warning" to
debugger and terminal as warning call
```

---

float **rad_to_deg**(rad: float) 🔗

Converts an angle expressed in radians to degrees.

```
rad_to_deg(0.523599) # Returns 30
rad_to_deg(PI)       # Returns 180
rad_to_deg(PI * 2)   # Returns 360
```

PackedInt64Array **rand_from_seed**(seed: int) 🔗

Given a `seed`, returns a PackedInt64Array of size `2`, where its first element is the randomized int value, and the second element is the same as `seed`. Passing the same `seed` consistently returns the same array.

Note: "Seed" here refers to the internal state of the pseudo random number generator, currently implemented as a 64 bit integer.

```
var a = rand_from_seed(4)

print(a[0]) # Prints 2879024997
print(a[1]) # Prints 4
```

---

float **randf**()

Returns a random floating-point value between `0.0` and `1.0` (inclusive).

GDScript

```
randf() # Returns e.g. 0.375671
```

C#

```
GD.Randf(); // Returns e.g. 0.375671
```

---

float **randf_range**(from: float, to: float)

Returns a random floating-point value between `from` and `to` (inclusive).

GDScript

```
randf_range(0, 20.5) # Returns e.g. 7.45315
randf_range(-10, 10) # Returns e.g. -3.844535
```

C#

```
GD.RandRange(0.0, 20.5);   // Returns e.g. 7.45315
GD.RandRange(-10.0, 10.0); // Returns e.g. -3.844535
```

---

float **randfn**(mean: float, deviation: float)

Returns a normally-distributed [https://en.wikipedia.org/wiki/Normal_distribution], pseudo-random floating-point value from the specified `mean` and a standard `deviation`. This is also known as a Gaussian distribution.

Note: This method uses the Box-Muller transform [https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform] algorithm.

---

int **randi**()

Returns a random unsigned 32-bit integer. Use remainder to obtain a random value in the interval `[0, N - 1]` (where N is smaller than 2^32).

GDScript

```
randi()           # Returns random integer between 0 and 2^32 - 1
randi() % 20      # Returns random integer between 0 and 19
randi() % 100     # Returns random integer between 0 and 99
randi() % 100 + 1 # Returns random integer between 1 and 100
```

C#

```
GD.Randi();           // Returns random integer between 0 and 2^32 - 1
GD.Randi() % 20;      // Returns random integer between 0 and 19
GD.Randi() % 100;     // Returns random integer between 0 and 99
GD.Randi() % 100 + 1; // Returns random integer between 1 and 100
```

---

int **randi_range**(from: int, to: int)

Returns a random signed 32-bit integer between `from` and `to` (inclusive). If `to` is lesser than `from`, they are swapped.

GDScript

```
randi_range(0, 1)       # Returns either 0 or 1
randi_range(-10, 1000) # Returns random integer between -10 and 1000
```

C#

```
GD.RandRange(0, 1);       // Returns either 0 or 1
GD.RandRange(-10, 1000); // Returns random integer between -10 and 1000
```

---

void **randomize**()

Randomizes the seed (or the internal state) of the random number generator. The current implementation uses a number based on the device's time.

Note: This function is called automatically when the project is run. If you need to fix the seed to have consistent, reproducible results, use seed to initialize the random number generator.

---

float **remap**(value: float, istart: float, istop: float, ostart: float, ostop: float)

Maps a `value` from range `[istart, istop]` to `[ostart, ostop]`. See also lerp and inverse_lerp. If `value` is outside `[istart, istop]`, then the resulting value will also be outside `[ostart, ostop]`. If this is not desired, use clamp on the result of this function.

```
remap(75, 0, 100, -1, 1) # Returns 0.5
```

For complex use cases where multiple ranges are needed, consider using Curve or Gradient instead.

Note: If `istart == istop`, the return value is undefined (most likely NaN, INF, or -INF).

---

int **rid_allocate_id**()

Allocates a unique ID which can be used by the implementation to construct an RID. This is used mainly from native extensions to implement servers.

RID **rid_from_int64**(base: int)

Creates an RID from a `base`. This is used mainly from native extensions to build servers.

---

float **rotate_toward**(from: float, to: float, delta: float)

Rotates `from` toward `to` by the `delta` amount. Will not go past `to`.

Similar to move_toward, but interpolates correctly when the angles wrap around @GDScript.TAU.

If `delta` is negative, this function will rotate away from `to`, toward the opposite angle, and will not go past the opposite angle.

Variant **round**(x: Variant)

Rounds `x` to the nearest whole number, with halfway cases rounded away from 0. Supported types: int, float, Vector2, Vector2i, Vector3, Vector3i, Vector4, Vector4i.

```
round(2.4) # Returns 2
round(2.5) # Returns 3
round(2.6) # Returns 3
```

See also floor, ceil, and snapped.

Note: For better type safety, use roundf, roundi, Vector2.round, Vector3.round, or Vector4.round.

---

float **roundf**(x: float)

Rounds `x` to the nearest whole number, with halfway cases rounded away from 0.

A type-safe version of round, returning a float.

---

int **roundi**(x: float)

Rounds `x` to the nearest whole number, with halfway cases rounded away from 0.

A type-safe version of round, returning an int.

---

void **seed**(base: int)

Sets the seed for the random number generator to `base`. Setting the seed manually can ensure consistent, repeatable results for most random functions.

GDScript

```
var my_seed = "Godot Rocks".hash()
seed(my_seed)
var a = randf() + randi()
seed(my_seed)
```

```
var b = randf() + randi()
# a and b are now identical
```

C#

```
ulong mySeed = (ulong)GD.Hash("Godot Rocks");
GD.Seed(mySeed);
var a = GD.Randf() + GD.Randi();
GD.Seed(mySeed);
var b = GD.Randf() + GD.Randi();
// a and b are now identical
```

Variant **sign**(x: Variant)

Returns the same type of Variant as `x`, with `-1` for negative values, `1` for positive values, and `0` for zeros. For `nan` values it returns 0.

Supported types: int, float, Vector2, Vector2i, Vector3, Vector3i, Vector4, Vector4i.

```
sign(-6.0) # Returns -1
sign(0.0)  # Returns 0
sign(6.0)  # Returns 1
sign(NAN)  # Returns 0

sign(Vector3(-6.0, 0.0, 6.0)) # Returns (-1, 0, 1)
```

Note: For better type safety, use signf, signi, Vector2.sign, Vector2i.sign, Vector3.sign, Vector3i.sign, Vector4.sign, or Vector4i.sign.

---

float **signf**(x: float)

Returns `-1.0` if `x` is negative, `1.0` if `x` is positive, and `0.0` if `x` is zero. For `nan` values of `x` it returns 0.0.

```
signf(-6.5) # Returns -1.0
signf(0.0)  # Returns 0.0
signf(6.5)  # Returns 1.0
signf(NAN)  # Returns 0.0
```

---

int **signi**(x: int)

Returns `-1` if `x` is negative, `1` if `x` is positive, and `0` if if `x` is zero.

```
signi(-6) # Returns -1
signi(0)  # Returns 0
signi(6)  # Returns 1
```

---

float **sin**(angle_rad: float)

Returns the sine of angle `angle_rad` in radians.

```
sin(0.523599)       # Returns 0.5
sin(deg_to_rad(90)) # Returns 1.0
```

---

float **sinh**(x: float)

Returns the hyperbolic sine of `x`.

```
var a = log(2.0) # Returns 0.693147
sinh(a) # Returns 0.75
```

---

float **smoothstep**(from: float, to: float, x: float)

Returns the result of smoothly interpolating the value of `x` between `0` and `1`, based on the where `x` lies with respect to the edges `from` and `to`.

The return value is `0` if `x <= from`, and `1` if `x >= to`. If `x` lies between `from` and `to`, the returned value follows an S-shaped curve that maps `x` between `0` and `1`.

This S-shaped curve is the cubic Hermite interpolator, given by `f(y) = 3*y^2 - 2*y^3` where `y = (x-from) / (to-from)`.

```
smoothstep(0, 2, -5.0) # Returns 0.0
smoothstep(0, 2, 0.5) # Returns 0.15625
smoothstep(0, 2, 1.0) # Returns 0.5
smoothstep(0, 2, 2.0) # Returns 1.0
```

Compared to ease with a curve value of `-1.6521`, smoothstep returns the smoothest possible curve with no sudden changes in the derivative. If you need to perform more advanced transitions, use Tween or AnimationPlayer.

Comparison between smoothstep() and ease(x, -1.6521) return values [https://raw.githubusercontent.com/godotengine/godot-docs/master/img/smoothstep_ease_comparison.png]

Variant **snapped**(x: Variant, step: Variant) 🔗

Returns the multiple of `step` that is the closest to `x`. This can also be used to round a floating-point number to an arbitrary number of decimals.

The returned value is the same type of Variant as `step`. Supported types: int, float, Vector2, Vector2i, Vector3, Vector3i, Vector4, Vector4i.

```
snapped(100, 32)  # Returns 96
snapped(3.14159, 0.01)  # Returns 3.14

snapped(Vector2(34, 70), Vector2(8, 8))  # Returns (32, 72)
```

See also ceil, floor, and round.

Note: For better type safety, use snappedf, snappedi, Vector2.snapped, Vector2i.snapped, Vector3.snapped, Vector3i.snapped, Vector4.snapped, or Vector4i.snapped.

---

float **snappedf**(x: float, step: float)

Returns the multiple of `step` that is the closest to `x`. This can also be used to round a floating-point number to an arbitrary number of decimals.

A type-safe version of snapped, returning a float.

```
snappedf(32.0, 2.5)   # Returns 32.5
snappedf(3.14159, 0.01)   # Returns 3.14
```

---

int **snappedi**(x: float, step: int)

Returns the multiple of `step` that is the closest to `x`.

A type-safe version of snapped, returning an int.

```
snappedi(53, 16)   # Returns 48
snappedi(4096, 100)   # Returns 4100
```

---

float **sqrt**(x: float)

Returns the square root of `x`, where `x` is a non-negative number.

```
sqrt(9)      # Returns 3
sqrt(10.24) # Returns 3.2
sqrt(-1)     # Returns NaN
```

Note: Negative values of `x` return NaN ("Not a Number"). in C#, if you need negative inputs, use `System.Numerics.Complex`.

int **step_decimals**(x: float)

Returns the position of the first non-zero digit, after the decimal point. Note that the maximum return value is 10, which is a design decision in the implementation.

```
var n = step_decimals(5)           # n is 0
n = step_decimals(1.0005)          # n is 4
n = step_decimals(0.000000005) # n is 9
```

String **str**(...) vararg

Converts one or more arguments of any Variant type to a String in the best way possible.

```
var a = [10, 20, 30]
var b = str(a)
print(len(a)) # Prints 3 (the number of elements in the array).
print(len(b)) # Prints 12 (the length of the string "[10, 20,
30]").
```

Variant **str_to_var**(string: String)

Converts a formatted `string` that was returned by var_to_str to the original Variant.

GDScript

```
var data = '{ "a": 1, "b": 2 }' # data is a String
var dict = str_to_var(data)      # dict is a Dictionary
print(dict["a"])                 # Prints 1
```

C#

```
string data = "{ \"a\": 1, \"b\": 2 }";          // data is a
string
var dict = GD.StrToVar(data).AsGodotDictionary(); // dict is a
```

```
Dictionary
GD.Print(dict["a"]);                                  // Prints 1
```

float **tan**(angle_rad: float) 🔗

Returns the tangent of angle `angle_rad` in radians.

```
tan(deg_to_rad(45)) # Returns 1
```

float **tanh**(x: float) 🔗

Returns the hyperbolic tangent of `x`.

```
var a = log(2.0) # Returns 0.693147
tanh(a)          # Returns 0.6
```

Variant **type_convert**(variant: Variant, type: int) 🔗

Converts the given `variant` to the given `type`, using the Variant.Type values. This method is generous with how it handles types, it can automatically convert between array types, convert numeric Strings to int, and converting most things to String.

If the type conversion cannot be done, this method will return the default value for that type, for example converting Rect2 to Vector2 will always return Vector2.ZERO. This method will never show error messages as long as `type` is a valid Variant type.

The returned value is a Variant, but the data inside and its type will be the same as the requested type.

```
type_convert("Hi!", TYPE_INT) # Returns 0
type_convert("123", TYPE_INT) # Returns 123
type_convert(123.4, TYPE_INT) # Returns 123
type_convert(5, TYPE_VECTOR2) # Returns (0, 0)
type_convert("Hi!", TYPE_NIL) # Returns null
```

String **type_string**(type: int) 🔗

Returns a human-readable name of the given `type`, using the Variant.Type values.

```
print(TYPE_INT) # Prints 2.
print(type_string(TYPE_INT)) # Prints "int".
print(type_string(TYPE_STRING)) # Prints "String".
```

See also typeof.

---

int **typeof**(variable: Variant) 🔗

Returns the internal type of the given `variable`, using the Variant.Type values.

```
var json = JSON.new()
json.parse('["a", "b", "c"]')
var result = json.get_data()
if typeof(result) == TYPE_ARRAY:
    print(result[0]) # Prints a
else:
    print("Unexpected result")
```

See also type_string.

PackedByteArray **var_to_bytes**(variable: Variant) 🔗

Encodes a Variant value to a byte array, without encoding objects. Deserialization can be done with bytes_to_var.

Note: If you need object serialization, see var_to_bytes_with_objects.

Note: Encoding Callable is not supported and will result in an empty value, regardless of the data.

PackedByteArray **var_to_bytes_with_objects**(variable: Variant) 🔗

Encodes a Variant value to a byte array. Encoding objects is allowed (and can potentially include executable code). Deserialization can be done with bytes_to_var_with_objects.

Note: Encoding Callable is not supported and will result in an empty value, regardless of the data.

String **var_to_str**(variable: Variant) 🔗

Converts a Variant `variable` to a formatted String that can then be parsed using str_to_var.

GDScript

```
var a = { "a": 1, "b": 2 }
print(var_to_str(a))
```

C#

```
var a = new Godot.Collections.Dictionary { ["a"] = 1, ["b"] = 2
};
GD.Print(GD.VarToStr(a));
```

Prints:

```
{
    "a": 1,
    "b": 2
}
```

Note: Converting Signal or Callable is not supported and will result in an empty value for these types, regardless of their data.

Variant **weakref**(obj: Variant)

Returns a WeakRef instance holding a weak reference to `obj`. Returns an empty WeakRef instance if `obj` is `null`. Prints an error and returns `null` if `obj` is neither Object-derived nor `null`.

A weak reference to an object is not enough to keep the object alive: when the only remaining references to a referent are weak references, garbage collection is free to destroy the referent and reuse its memory for something else. However, until the object is actually destroyed the weak reference may return the object even if there are no strong references to it.

Variant **wrap**(value: Variant, min: Variant, max: Variant)

Wraps the Variant `value` between `min` and `max`. Can be used for creating loop-alike behavior or infinite surfaces.

Variant types int and float are supported. If any of the arguments is float this function returns a float, otherwise it returns an int.

```
var a = wrap(4, 5, 10)
# a is 9 (int)

var a = wrap(7, 5, 10)
# a is 7 (int)

var a = wrap(10.5, 5, 10)
# a is 5.5 (float)
```

---

float **wrapf**(value: float, min: float, max: float)

Wraps the float `value` between `min` and `max`. Can be used for creating loop-alike behavior or infinite surfaces.

```
# Infinite loop between 5.0 and 9.9
value = wrapf(value + 0.1, 5.0, 10.0)
```

```
# Infinite rotation (in radians)
angle = wrapf(angle + 0.1, 0.0, TAU)
```

```
# Infinite rotation (in radians)
angle = wrapf(angle + 0.1, -PI, PI)
```

Note: If `min` is `0`, this is equivalent to fposmod, so prefer using that instead.

wrapf is more flexible than using the fposmod approach by giving the user control over the minimum value.

---

int **wrapi**(value: int, min: int, max: int)

Wraps the integer `value` between `min` and `max`. Can be used for creating loop-alike behavior or infinite surfaces.

```
# Infinite loop between 5 and 9
frame = wrapi(frame + 1, 5, 10)
```

```
# result is -2
var result = wrapi(-6, -5, -1)
```